МАГІСТЕРСЬКА РОБОТА

МР. ІПм - 55.00.00.000 ПЗ

Група ІПм-23-2

Павлишин Владислав

2024

Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

Павлишин Владислав Михайлович

(прізвище, ім'я, по батькові)

УДК 004.9

(індекс)

МАГІСТЕРСЬКА РОБОТА

Методи та засоби управління додатком на основі оцінок продуктивності

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121 - Інженерія програмного забезпечення

(шифр і назва спеціальності)

Павлишин В.М.

(підпис, ініціали та прізвище здобувача освітнього ступеня)

Науковий керівник Піх Володимир Ярославович, к.т.н., доцент

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Допущено до захисту

Завідувач кафедри

доц. Бандура В.В.

(посада) (підпис) (дата) (ініціали та прізвище)

Нормоконтроль

доц. Вовк Р.Б.

(посада) (підпис) (дата) (ініціали та прізвище)

Робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

Івано-Франківськ – 2024

Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

Освітній рівень магістр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою **ІПЗ**
доц. В.В. Бандура
“ 04 ” вересня 2024 р.

ЗАВДАННЯ

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Павлишину Владиславу Михайловичу

(прізвище, ім’я, по-батькові)

1. Тема магістерської роботи “ Методи та засоби управління додатком на основі оцінок продуктивності ”

керівник проекту (роботи) Піх Володимир Ярославович, к.т.н., доцент

затверджені наказом закладу вищої освіти від “ 22 ” листопада 2024 р. № 781/7

2. Строк подання студентом проекту (роботи) 15 грудня 2024 р.

3. Вихідні дані до проекту (роботи) Теоретичні концепції та формальні моделі побудови та функціонування інформаційних технологій оцінки продуктивності

4. Зміст розрахунково - пояснювальної записки(перелік питань, які потрібно розробити)

1. Дослідження проблеми прогнозування в заданій предметній області
2. Аналіз алгоритмів машинного навчання процесів прогнозування
3. Методи, моделі та засоби управління додатком на основі оцінок продуктивності
4. Імплементація методів та алгоритмів управління додатком на основі оцінок продуктивності

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

1. Протокол тестування (рис. 1.1)
2. Огляд штучної нейронної мережі (рис. 1.2)
3. Класифікація за допомогою SVM (рис. 1.3)
4. Приклад дерева рішень (рис. 1.4)
5. Приклад поділу тестового набору в k-кратній перехресній перевірці (рис. 1.5)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата
Перевірка на плагіат	доц., к.т.н. Вовк Р.Б.	

7. Дата видачі завдання 04 вересня 2024 р.

Керівник _____
(підпис)

Завдання прийняв до виконання_____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назви етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1	Підбір і вивчення літератури по темі магістерської роботи	15.09.2024	виконано
2	Аналіз концепцій та алгоритмів предметної області	29.09.2024	виконано
3	Дослідження проблеми прогнозування в заданій предметній області	15.10.2024	виконано
4	Аналіз алгоритмів машинного навчання процесів прогнозування	08.11.2024	виконано
5	Методи, моделі та засоби управління додатком на основі оцінок продуктивності	20.11.2024	виконано
6	Імплементація методів та алгоритмів управління додатком на основі оцінок продуктивності	01.12.2024	виконано
7	Затвердження пояснювальної записки роботи завідувачем кафедри	15.12.2024	виконано

Студент – магістр _____
(підпис)

Керівник роботи _____
(підпис)

АНОТАЦІЯ

Магістерська робота: 75 с., 20 рис., 5 табл., 49 джерел.

Тема: Методи та засоби управління додатком на основі оцінок продуктивності

Об'єкт дослідження: процеси прогнозування та управління інформаційними системами на основі методів машинного навчання.

Мета роботи: дослідження методів, моделей і засобів управління додатком на основі оцінок продуктивності алгоритмів машинного навчання для підвищення точності прогнозування в заданій предметній області.

Предмет дослідження: методи, моделі та алгоритми машинного навчання, що застосовуються для управління додатком і прогнозування продуктивності.

Результати дослідження

В роботі запропоновано нові підходи до оцінки продуктивності алгоритмів машинного навчання, які дозволяють підвищити точність та інтерпретованість результатів.

Висновок

Прогнозування на основі оцінок продуктивності дозволило отримати високоточні передбачення, які можуть бути застосовані для підтримки прийняття рішень у різних сценаріях.

ПРОГНОЗУВАННЯ, МАШИННЕ НАВЧАННЯ, АЛГОРИТМИ, ПРОДУКТИВНІСТЬ, УПРАВЛІННЯ ДОДАТКОМ, ВЕБ-СЕРВІС, БІБЛІОТЕКА SCIKIT-LEARN, NUMPY.

ABSTRACT

Master Thesis: 75 pp., 20 fig., 5 tab., 49 sources.

Thesis Subject: Methods and tools for application management based on performance assessments

Object of research: forecasting processes and information systems management based on machine learning methods.

Purpose of work: study of methods, models and tools for application management based on performance assessments of machine learning algorithms to increase the accuracy of forecasting in a given subject area.

Subject of research: methods, models and algorithms of machine learning used for application management and performance forecasting.

Research results

The paper proposes new approaches to assessing the performance of machine learning algorithms, which allow to increase the accuracy and interpretability of results.

Conclusion

Forecasting based on performance assessments allowed to obtain highly accurate predictions that can be used to support decision-making in various scenarios.

FORECASTING, MACHINE LEARNING, ALGORITHMS, PRODUCTIVITY, APPLICATION MANAGEMENT, WEB SERVICE, SCIKIT-LEARN LIBRARY, NUMPY.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ML - Machine Learning

ORM - Object Relational Mapping

REST - Representational State Transfer

CRUD REST operations: Create, Read, Update, Delete

SVM - Support Vector Machine

ANN - Artificial Neural Network

SSL - Secure Socket Layer

ВСТУП

Актуальність теми.

Зростаючий обсяг даних та складність сучасних інформаційних систем обумовлюють необхідність впровадження нових технологій для обробки, аналізу та прогнозування. Алгоритми машинного навчання стали ключовим інструментом для створення адаптивних і точних моделей, здатних забезпечувати ефективне управління системами у складних та змінних умовах.

У багатьох предметних областях, таких як бізнес, фінанси, медицина, логістика, управління ресурсами тощо, важливість точного прогнозування постійно зростає. Це зумовлено потребою в оптимізації процесів, скороченні витрат, підвищенні якості послуг і продуктів. Проте основними викликами залишаються висока варіативність даних, складність моделей та обмежені обчислювальні ресурси, що потребують спеціалізованих підходів для досягнення необхідної продуктивності.

Особливої актуальності набуває інтеграція прогнозування в мобільні додатки та веб-сервіси, які сьогодні є основним інструментом для взаємодії користувачів із системами. Створення таких рішень потребує розробки моделей і алгоритмів, що є не лише високоточними, а й ресурсозберігаючими, щоб забезпечити швидкість і зручність роботи з додатком.

Крім того, у сучасному середовищі важливою вимогою є забезпечення гнучкості систем для адаптації до змінюваних умов. Це потребує використання багатофакторного підходу, який враховує взаємодію численних характеристик, таких як якість вхідних даних, швидкість обчислень, рівень інтеграції з існуючими інструментами та вимоги кінцевих користувачів.

Таким чином, дослідження, спрямовані на створення ефективних методів прогнозування та управління інформаційними системами з

використанням алгоритмів машинного навчання, мають як теоретичну, так і практичну значущість. Розробка інтегрованих програмних рішень, які забезпечують якісне прогнозування та оптимізацію процесів, сприятиме вирішенню низки актуальних проблем у багатьох галузях.

Мета дослідження – дослідження методів, моделей і засобів управління додатком на основі оцінок продуктивності алгоритмів машинного навчання для підвищення точності прогнозування в заданій предметній області.

Об'єкт дослідження - процеси прогнозування та управління інформаційними системами на основі методів машинного навчання.

Предмет дослідження - методи, моделі та алгоритми машинного навчання, що застосовуються для управління додатком і прогнозування продуктивності.

Задачі дослідження

1. Провести аналіз сучасних проблем прогнозування у заданій предметній області.

2. Дослідити фактори та характеристики, що впливають на продуктивність алгоритмів машинного навчання.

3. Вивчити основні алгоритми машинного навчання, їхні переваги та обмеження для застосування у прогнозуванні.

4. Розробити методологію управління додатком, що включає підготовку даних, налаштування алгоритмів та оцінку продуктивності.

5. Реалізувати інтегровану систему, яка включає веб-сервіс і мобільний додаток для управління даними та виконання прогнозування.

Методи дослідження:

- Аналітичні методи для вивчення літератури та визначення проблеми.

- Методи машинного навчання для побудови моделей прогнозування.

- Емпіричні методи для тестування алгоритмів на реальних даних.

- Інструменти програмного забезпечення, зокрема бібліотеки scikit-learn і NumPy, для реалізації та оцінки алгоритмів.

Наукова новизна отриманих результатів

Удосконалено підходи до прогнозування продуктивності за допомогою алгоритмів машинного навчання, зокрема адаптацію їх до динамічних умов. Розроблено інтегровану систему управління додатком, що забезпечує зберігання, обробку та прогнозування даних у реальному часі.

Практичне значення результатів

Результати роботи можуть бути використані для розробки програмних рішень у галузі прогнозування, управління інформаційними системами та аналізу продуктивності. Інтегрована система з веб-сервісом і мобільним додатком може бути впроваджена в різних галузях, таких як бізнес-аналітика, управління ризиками, фінансове планування та інші.

Структура магістерської роботи. Робота складається зі вступу, трьох розділів та висновків. Загальний обсяг роботи становить 75 сторінок, і містить 20 рисунків, 5 таблиць, список використаних джерел із 49 найменувань, 1 додаток

РОЗДІЛ 1. ДОСЛІДЖЕННЯ ПРОБЛЕМИ ПРОГНОЗУВАННЯ В ЗАДАНІЙ ПРЕДМЕТНІЙ ОБЛАСТІ

1.1. Опис проблеми дослідження

У лижних гонках одним з важливих факторів для спортсменів високого рівня є наявність лиж з хорошими характеристиками тертя об сніг. Снігові умови можуть сильно відрізнятися між змаганнями, і різні пари лиж добре працюють залежно від цих умов.

Щоб відповідати різним сніговим умовам, кожен спортсмен збірної має 40-50 пар лиж. Для того, щоб визначити, яка пара буде найбільш придатною для конкретних змагань, кожну пару необхідно протестувати за різних умов, перш ніж можна буде зробити вибір на основі результатів.

Лижні пари кожного спортсмена готуються та тестуються сервісменами. Наразі, залежно від уподобань сервісмена, тестові сесії записуються у фізичні блокноти або електронні таблиці. Кожен сервісмен у збірній відповідає за лижі 3-4 спортсменів. Коли сервісмену потрібно вибрати лижі для спортсмена, за якого він не відповідає, це може бути проблематично, оскільки в команді немає загального обліку лиж та результатів тестів. Через ці труднощі національна збірна потребує рішення для зберігання інформації про лижі та тести лиж, а також для можливості використовувати цю інформацію для прогнозування характеристик пар лиж за заданих умов. Це рішення має на меті допомогти сервісменам у виборі найкращих лиж для змагань на основі їх попередніх результатів.

Через велику кількість лиж, задіяних у збірній, сервісменам важко відстежувати всі пари лиж, особливо тим, хто не має досвіду роботи з лижами певного спортсмена. Велика кількість лиж також означає, що не всі пари лиж спортсмена можуть брати участь у кожній тестовій сесії. Наразі національній збірній бракує добре функціонуючого способу вибору пар лиж з найвищою ймовірністю хорошого результату на основі попередніх

результатів тестів. Вимога збірної полягає в тому, щоб така система була надійною. Надійність може бути виражена як близькість до нуля випадків, коли фактично найкраща пара лиж прогнозується як така, що покаже погані результати. Низька надійність означатиме високий ризик того, що фактично найкраща пара може бути не обрана. Проблеми, які стоять на порядку денному, можна підсумувати такими питаннями:

- Як можна зібрати інформацію з тестових сесій для системи прогнозування, щоб передбачити оптимальні лижі за заданих умов?

- Як можна позначити зібрану інформацію для підтримки системи прогнозування?

- Використовуючи алгоритм машинного навчання, наскільки надійно можна передбачити оптимальні лижі за заданих умов?

- З різних алгоритмів з різними упередженнями, який алгоритм є найбільш придатним для проекту з точки зору точності та раніше визначеної надійності?

Метою цього проекту є допомога збірній у виборі лиж для змагань шляхом створення інструменту для прогнозування найкращих лиж за заданих умов. Буде створено рішення для управління лижами та тестовими сесіями. Інформація, що зберігається в цьому рішенні, стане основою для прогнозів.

Обсяг цього проекту полягає в прогнозуванні найкращих лиж за заданих умов за допомогою готових алгоритмів машинного навчання. Налаштування параметрів алгоритмів не входить в обсяг цього проекту. Крім того, час виконання різних алгоритмів машинного навчання не буде оцінюватися. Для зберігання та обміну інформацією буде реалізовано сервіс, що працює разом з системою баз даних. Система прогнозування буде реалізована з використанням цього онлайн-сервісу для цілей валідації в цьому проекті.

Варто зазначити, що не буде реалізовано систему рекомендацій з використанням прогнозів для кінцевого користувача. Деталі безпеки щодо зв'язку між онлайн-сервісом та мобільним застосунком не будуть

розглядатися. Проект буде використовувати результати тестів лиж, але практична реалізація тестів лиж та збір їх результатів не входять в обсяг проекту. Крім того, в прогнозах будуть враховуватися лише тести лиж, де лижі ранжуються на основі людської оцінки після тестування на різній місцевості. Тестування лижних мазей та різних шліфувань не буде включено.

1.2. Опис факторів та характеристик що впливають на продуктивність досліджуваної області

Цей розділ містить загальну інформацію про тестування лиж, яке проводить збірна, а також опис факторів, що впливають на характеристики бігових лиж. Крім того, надається інформація, що стосується рішення.

Як описується в [1], учасники лижних гонок докладають значних зусиль для зменшення тертя ковзання лиж. Тертя ковзання змінюється залежно від снігових умов. Снігові умови можна визначити за температурою, вологістю та типом снігу. Автор поділяє тип снігу на категорії: свіжий, старий та трансформований. Факторами, які, згідно [2], впливають на характеристики лиж, є структура їх поверхні, яку можна змінити шляхом шліфування, та різні форми нанесення мазей. Різні шліфування підходять для різних снігових умов. Хоча мазі також працюють по-різному залежно від снігових умов, вони не будуть розглядатися в цьому проекті. Автор в [3] стверджує, що на характеристики лиж впливає крива прогину та розподіл тиску на сніг. Один тип кривої прогину може добре працювати в одних умовах, але може не підходити в інших. Крім того, сервісмен з підготовки лиж описав проблеми в [4], пов'язані з варіаціями характеристик лиж в різних умовах та через невизначеність виробництва лиж. Виробники лиж не можуть виготовити кілька лиж з однаковими властивостями. Таким чином, може бути важко оцінити, чи буде пара лиж добре працювати в певних умовах без тестування. Одна пара лиж може перевершувати іншу пару в певних умовах, але в інших умовах ситуація може бути зворотною.

Щоб впоратися з різними сніговими умовами, кожен спортсмен має 40-50 пар лиж, і виробники лиж постійно надають додаткові лижі. Збірна повинна оцінити всі ці лижі в різних умовах. Лижі оцінюються шляхом польових випробувань, де одночасно тестуються дві пари. Під час тестування ковзанярських лиж, де вся ковзна поверхня лиж підготовлена для оптимального ковзання, два тестувальники порівнюють дві пари лиж одночасно, ковзаючи паралельно один одному вниз по схилу. Під час тестування лиж для класичного стилю, де на середню частину лиж наноситься мазь зчеплення, щоб забезпечити діагональний підйом в гору, процедура тестування відрізняється. Сервісмени оцінюють лижі на різній місцевості та ранжують їх на основі їх загальних характеристик як на підйомах, так і на рівнинній місцевості та спусках. У всіх тестових сесіях, що стосуються цього проекту, всі лижі підготовлені однаковою маззю, але шліфування ковзної поверхні може відрізнятися. Лижі завжди розглядаються парами в ситуаціях, що стосуються цього проекту, незважаючи на те, що всі лижі унікальні. Кожна лижа в парі позначена відповідним номером.

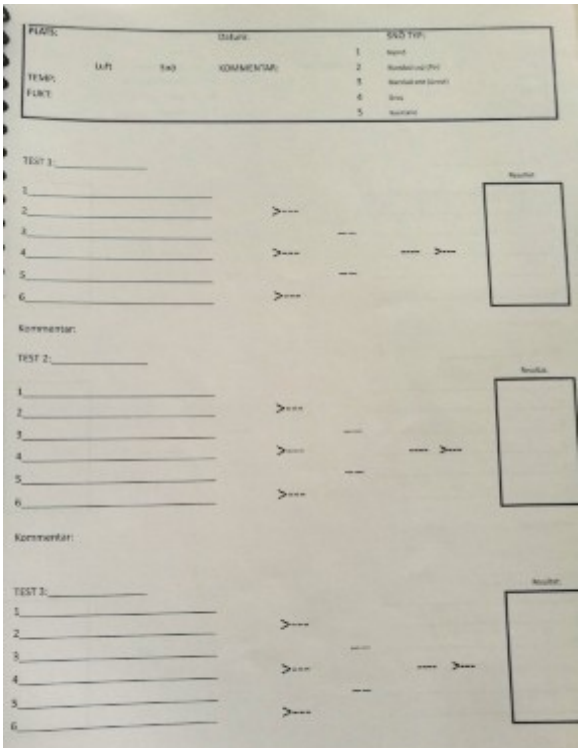


Рис. 1.1. Протокол тестування

Для ведення обліку польових випробувань тестові сесії наразі записуються у фізичні блокноти або електронні таблиці залежно від уподобань сервісмена. На рисунку 1.1 показано приклад протоколу, який використовується для тестових сесій. Тести лиж перед змаганнями проводяться у два етапи. За день до змагань відбирається та тестується певна кількість пар лиж. На основі результатів цих тестів, як правило, від 2 до 5 з цих пар лиж відбираються для тестів у день змагань. Цей проект зосереджений на тестах, що проводяться за день до змагань. Визначаючи, які лижі тестувати за день до змагань, сервісмени використовують свій досвід шліфування та лиж в очікуваних снігових умовах, щоб зробити свій вибір. Відібрані лижі тестуються одна проти одної, і результат вноситься в протокол. Процедуру відбору можна вважати схильною до людських помилок, оскільки сервісменам важко відстежувати всі доступні лижі. Крім того, якщо пара лиж показала надзвичайно хороші результати на минулих змаганнях, лижникам та сервісменам важко не віддати перевагу цій парі.

Кожен сервісмен несе основну відповідальність за лижі обмеженої кількості спортсменів національної збірної, як правило, одного чоловіка та однієї жінки. Під час деяких змагань сервісмен може бути у відпустці. У цих ситуаціях інший сервісмен повинен взяти на себе відповідальність. Замінюючому сервісмену може бути важко вибрати найкращі лижі, оскільки немає загальної системи для відстеження пар лиж кожного спортсмена.

1.3. Аналіз алгоритмів машинного навчання процесів прогнозування

Машинне навчання (ML) може бути використано для отримання прогнозу того, які пари лиж можуть працювати краще на основі попередніх даних. Згідно [4], прогнозування чисел, таких як температура завтрашнього дня, називається регресією, а прогнозування скінченного набору значень, таких як сонячно, хмарно або дощ, називається класифікацією. Для цього

проекту алгоритми класифікації можна вважати найбільш придатними, оскільки прогнози будуть в основному базуватися на мітках. У наступному розділі згадуються як регресійні, так і класифікаційні алгоритми на основі попередніх досліджень у цій галузі.

В дослідженні [5] припускають, що алгоритми ML можна використовувати для прогнозування спортивних результатів, і згадує ряд пов'язаних досліджень. Інші дослідники, зокрема в [6], також висловлювали подібні припущення. Серед досліджень, згаданих тут це штучна нейронна мережа (ANN) та інші види регресії використовувалися як алгоритми, а також алгоритми класифікації. Автор стверджує, що прогнозування спортивних результатів зазвичай розглядається як задача класифікації, де потрібно передбачити клас, такий як перемога, нічия або поразка. В [7] автор порівняв метод опорних векторів (SVM), ANN та дерево рішень для прогнозування результатів футбольних матчів коледжів. Всі три алгоритми передбачали результат ігор з точністю від 75% до 87%.

1.3.1. Алгоритми машинного навчання

В [4] описують три типи алгоритмів ML. Один тип ML називається навчанням з учителем, яке описується як таке, де і вхідні, і вихідні дані відомі в навчальному наборі. У наступному розділі представлено три різних алгоритми, засновані на навчанні з учителем: ANN, SVM та дерево рішень. Всі вони можуть вважатися придатними для прогнозування спортивних результатів, виходячи з досліджень у цій галузі, згаданих раніше. Крім того, представлено алгоритм, призначений для використання як базовий для порівняння.

1.3.2. Штучна нейронна мережа

В роботі [4] автор описує ШНМ як алгоритм нелінійної регресії, який імітує нейрони та зв’язки між ними в мозку. Кожне з’єднання має пов’язану з ним вагу, яка визначає, який вплив це з’єднання має мати на ШНМ. Щоб

налаштувати ці ваги, можна використовувати зворотне поширення. Це ітераційний процес, коли модель запускається кілька разів у навчальному наборі. На кожній ітерації ваги коригуються, щоб створити модель, яка якомога краще відповідає тестовому набору.

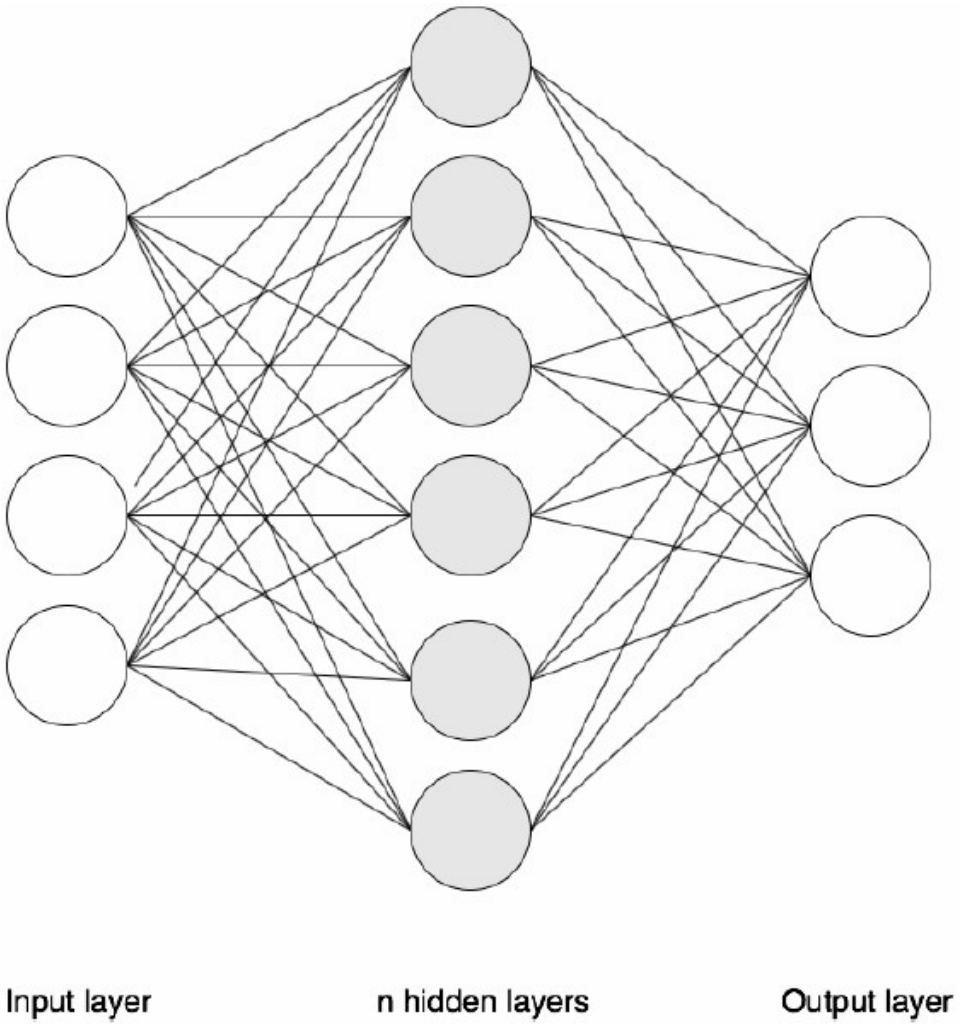


Рис. 1.2. Огляд штучної нейронної мережі

У ШНМ, відповідно до [8], модель, що складається з вхідного рівня, ряду прихованих шарів і вихідного рівня, визначені, як показано на рисунку 1.1. Немає особливого правила, як визначити оптимальну кількість прихованих шарів для поточної проблеми. Кожен шар складається з ряду нейронів, які є блоками обробки ШНМ. Для отримання виходу від кожного нейрона вказується функція активації. Важливим фактором у моделі є

кількість нейронів у прихованих шарах. Немає жодної гарантії, що збільшення цього числа покращить роботу алгоритму. Забагато нейронів може призвести до переобладнання. В [9] автор описує переобладнання як випадки, коли шум у навчальних даних враховується в моделі замість пошуку загальних правил прогнозування.

1.3.3. Метод опорних векторів (Support Vector Machine)

Згідно з [4], метод опорних векторів (SVM) - це алгоритм класифікації, який будує роздільник з максимальним відступом. Роздільник створює межу рішення з максимально можливою відстанню до точок вибірки.

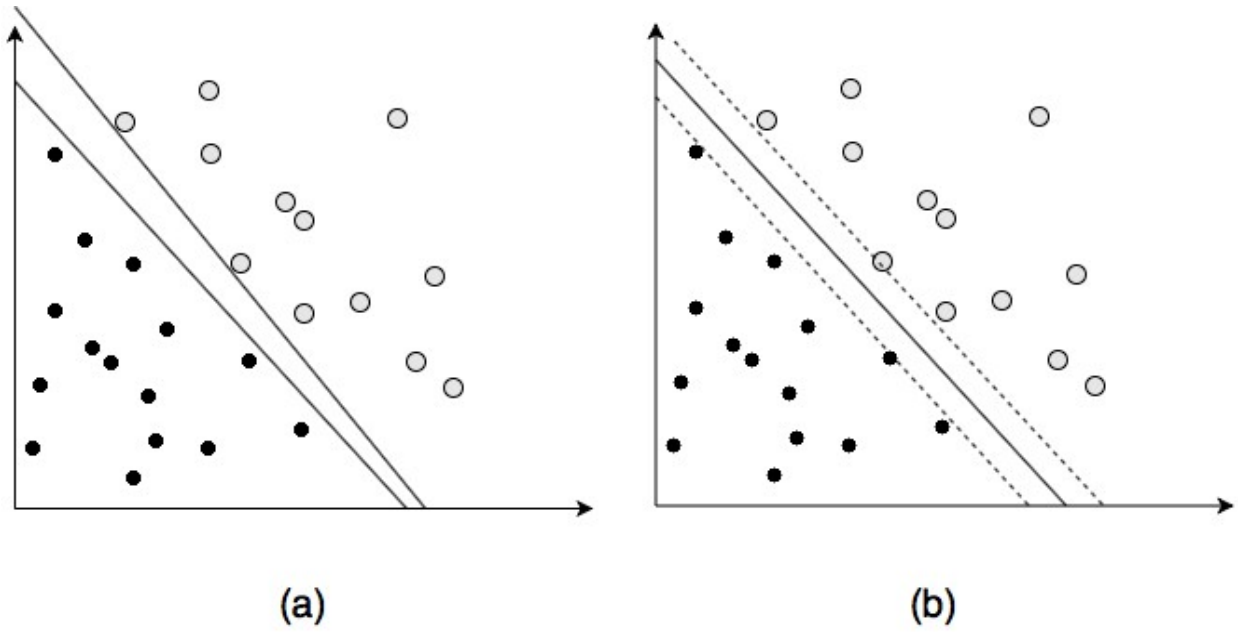
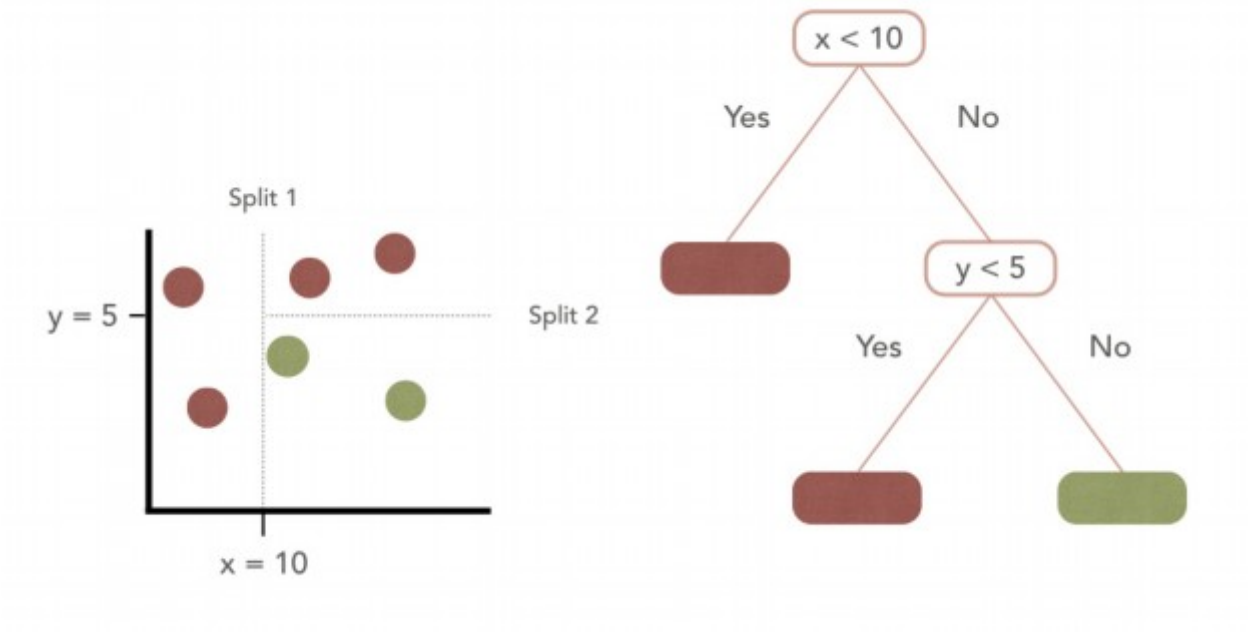


Рис. 1.3. Класифікація за допомогою SVM

Рисунок 1.3 показує класифікацію за допомогою SVM. Чорні та білі точки представляють два класи. На (a) дві можливі лінійні роздільники намальовані чорними лініями. На (b) чорна лінія представляє роздільник з максимальним відступом, а пунктирні лінії - це відступи. Якщо класифікувати точки даних між лінійними роздільниками на (a), класифікація, швидше за все, завершиться невдачею, використовуючи один з

цих роздільників, ніж якщо використовувати роздільник з максимальним відступом на (b). Якщо, наприклад, вибрати крайній правий роздільник на (a), точка даних ліворуч від крайньої лівої існуючої точки даних буде класифікована як чорна точка, незважаючи на її ближчу відстань до білої точки даних. Використовуючи роздільник з максимальним відступом на (b), ця проблема вирішується.

SVM використовує лінійне розділення даних, але якщо дані не є лінійно роздільними у двовимірному просторі, їх можна вбудувати у простір вищої розмірності. У цьому просторі створюється лінійна розділяюча гіперплощина, а результат потім переноситься назад у двовимірну площину. Згідно [4], якщо дані відображені на достатньо високу розмірність, майже всі дані можна лінійно розділити.



а) б)

Рис. 1.4. Приклад дерева рішень

1.3.4. Дерево рішень (Decision Tree)

В [4] описується дерево рішень як функцію, яка приймає значення атрибутів як вхідні дані та повертає одне вихідне значення. Функція дерева рішень виконує послідовність тестів на вхідних даних, щоб визначити

вихідне значення. Коли функція навчається, створюється дерево рішень з вузлами, кожен з яких відповідає тестам одного з вхідних значень. Потім створюються гілки з можливими значеннями атрибута. Кожен листовий вузол визначає повернене значення функції. На рисунку 1.4 показано приклад дерева рішень (б), створеного з набору даних, представленого на (а). Дані спочатку розділяються при x = 10. Оскільки є лише червоні точки, коли x < 10, класифікація може позначити дані як червоні, якщо x < 10. Якщо x >= 10, потрібен інший поділ для класифікації даних, тому створюється інший вузол з новою умовою. Звідси всі доступні дані можна класифікувати без подальшого розгалуження.

Алгоритм "Дерево рішень" - це популярний метод машинного навчання, який використовується для класифікації та регресії. Він будує модель у формі дерева, де кожен вузол представляє собою тест на атрибут, кожна гілка - результат тесту, а кожен лист - клас або значення прогнозу.

Ось основні кроки алгоритму:

- Вибір найкращого атрибуту. Алгоритм починає з вибору найкращого атрибуту для розділення даних. Для цього використовуються різні метрики, такі як ентропія, індекс Гіні або коефіцієнт приросту інформації. Мета - знайти атрибут, який найкраще розділяє дані на однорідні групи.

- Розбиття даних. Після вибору найкращого атрибуту дані розбиваються на підмножини на основі значень цього атрибуту.

- Рекурсивне побудова дерева. Для кожної підмножини даних кроки 1 і 2 повторюються рекурсивно, доки не буде досягнуто певного критерію зупинки. Це може бути максимальна глибина дерева, мінімальна кількість прикладів у вузлі або досягнення певного рівня чистоти вузла.

- Прогнозування. Для класифікації нового прикладу, починаючи з кореневого вузла, виконуються тести на атрибути, слідуючи гілкам дерева, доки не буде досягнуто листа. Клас або значення, пов'язане з цим листом, є прогнозом для даного прикладу.

Існують різні методи для покращення продуктивності дерев рішень, такі як:

- Обрізка. Видалення гілок дерева, які не вносять значного внеску в точність прогнозування.

- Ансамблі. Об'єднання кількох дерев рішень для створення більш точної та стабільної моделі, наприклад, випадковий ліс або градієнтний бустинг.

1.3.5. ZeroR - Базовий алгоритм

Результат алгоритмів машинного навчання можна порівняти з базовим рівнем, згенерованим алгоритмом ZeroR. ZeroR, за даними [13], - це алгоритм класифікації, який просто вибирає клас більшості.

ZeroR просто передбачає найпоширеніший клас у навчальному наборі даних. Іншими словами, він ігнорує всі атрибути або ознаки даних і просто вибирає клас, який зустрічається найчастіше.

Використовується ZeroR в наступних варіаціях:

- Базовий рівень. ZeroR встановлює базовий рівень продуктивності, з яким можна порівнювати інші алгоритми машинного навчання. Якщо складніший алгоритм не може перевершити ZeroR, це означає, що він не вивчає нічого корисного з даних.

- Оцінка даних. ZeroR може допомогти оцінити, наскільки добре дані піддаються класифікації. Якщо ZeroR має високу точність, це може означати, що класи в даних сильно незбалансовані, або що атрибути не є інформативними для прогнозування цільового атрибута.

ZeroR - це дуже простий алгоритм з очевидними обмеженнями. Він не враховує жодної інформації про атрибути даних і не може вивчати складні закономірності. Тому він рідко використовується для реальних завдань класифікації. Однак, як базовий рівень та інструмент оцінки даних, ZeroR може бути корисним.

Висновки до розділу

В даному розділі проведено комплексний аналіз проблеми прогнозування в заданій предметній області, що є основою для формування ефективних рішень та вибору оптимальних методів обробки даних. Опис проблеми дослідження дозволив ідентифікувати ключові аспекти, що впливають на точність і ефективність прогнозування. Серед них варто відзначити високу динамічність зовнішнього середовища, багатофакторність процесів, що вивчаються, та наявність невизначеностей, які ускладнюють побудову моделей.

Фактори та характеристики, що впливають на продуктивність систем у заданій предметній області, були детально розглянуті. З'ясовано, що точність прогнозування значною мірою залежить від обсягу, якості даних, параметрів алгоритмів, а також правильності попередньої обробки інформації. Залежність результату від обраного методу також була визначена як ключовий елемент для оптимізації процесу.

Аналіз алгоритмів машинного навчання показав, що існує широкий спектр підходів до вирішення задач прогнозування, кожен із яких має свої переваги та обмеження:

- Штучна нейронна мережа (Artificial Neural Network) виявилася потужним інструментом для розв'язання нелінійних задач, особливо у випадках складних взаємозв'язків між даними.

- Метод опорних векторів (SVM) забезпечує високу точність для невеликих наборів даних, особливо у випадках чіткої межі між класами.

- Дерево рішень (Decision Tree) є інтуїтивно зрозумілим та придатним для інтерпретації, проте схильне до перенавчання на великих наборах даних.

- ZeroR, як базовий алгоритм, використовується для оцінки базового рівня прогнозування, який виступає відправною точкою для оцінювання ефективності складніших підходів.

Результати цього розділу підкреслюють, що успішне прогнозування залежить від грамотного вибору алгоритму машинного навчання, налаштування його параметрів, а також ретельного аналізу вхідних даних. Далі дослідження буде спрямоване на порівняння ефективності методів і розробку оптимального підходу для вирішення завдань у конкретній предметній області.

РОЗДІЛ 2. МЕТОДИ, МОДЕЛІ ТА ЗАСОБИ УПРАВЛІННЯ ДОДАТКОМ НА ОСНОВІ ОЦІНОК ПРОДУКТИВНОСТІ

2.1. Методи оцінки продуктивності алгоритмів машинного навчання

Для порівняння різних алгоритмів машинного навчання необхідне вимірювання продуктивності. Два методи вимірювання продуктивності алгоритмів машинного навчання, які раніше використовувалися в спортивних прогнозах, - це точність (Accuracy) та збалансована точність (Balanced Accuracy). Згідно дослідження [7], точність - це кількість правильних прогнозів (істинно позитивні + істинно негативні), поділена на загальну кількість прогнозів (істинно позитивні + істинно негативні + хибно позитивні + хибно негативні):

$$\frac{TP+TN}{TP+TN+FP+FN}$$

Автор [7] використовував Accuracy, порівнюючи три різні алгоритми ML, застосовані до футбольних ігор коледжу. В [5] також згадується Accuracy у пропозиції структури, призначеної для прогнозування спортивних результатів. В [10] автор використовував Balanced Accuracy для вимірювання продуктивності різних алгоритмів ML, застосованих до ігор у крикет. Збалансована точність описана в [11] як спосіб представити справедливе вимірювання, коли набір даних є незбалансованим.

Balanced Accuracy (Збалансована точність) - це метрика, яка використовується в машинному навчанні для оцінки продуктивності моделей класифікації, особливо коли є дисбаланс класів (тобто, коли один клас представлений значно більше, ніж інші).

Звичайна точність (Accuracy) може бути оманливою при дисбалансі класів. Наприклад, якщо у вас є набір даних, де 90% прикладів належать до класу "A", а 10% - до класу "B", модель, яка просто передбачає "A" для всіх прикладів, матиме точність 90%. Однак, така модель не є корисною, оскільки вона не може правильно класифікувати приклади класу "B".

Balanced Accuracy вирішує цю проблему, обчислюючи середню точність для кожного класу окремо, а потім усереднюючи ці значення. Це означає, що кожен клас має однаковий внесок у загальну метрику, незалежно від його розміру.

$$Balanced\ Accuracy = (Sensitivity + Specificity) / 2$$

де:

- Sensitivity (Чутливість): частка правильно класифікованих позитивних прикладів (True Positive Rate).

- Specificity (Специфічність): частка правильно класифікованих негативних прикладів (True Negative Rate).

Дана метрика враховує дисбаланс класів, тобто Balanced Accuracy є більш надійною метрикою, ніж звичайна точність, коли є дисбаланс класів. Також вона надає збалансовану оцінку, а саме надає рівну вагу кожному класу, забезпечуючи більш справедливу оцінку продуктивності моделі.

Balanced Accuracy є важливою метрикою для оцінки моделей класифікації при дисбалансі класів. Вона забезпечує більш збалансовану та надійну оцінку продуктивності моделі, ніж звичайна точність.

Balanced Accuracy особливо корисна в таких задачах, як:

- Виявлення шахрайства, де шахрайські транзакції є рідкісними порівняно з легітимними.

- Медична діагностика, де виявлення рідкісних захворювань є критично важливим.

- Класифікація тексту, де деякі категорії можуть бути набагато менш поширеними, ніж інші.

Точність і збалансована точність вимірювань можна розрахувати за допомогою scikit-Learn.

В [12] зазначається, що оцінка алгоритму ML повинна бути зроблена на наборі даних, які він не бачив. Це можна вирішити шляхом випадкового розбиття набору даних на навчальний набір і тестовий набір. Оцінка проводиться шляхом порівняння прогнозованого результату набору тестів із фактичним результатом. Цей метод називається утримуваною перехресною перевіркою. Щоб отримати більше від даних, можна використати k-кратну перехресну перевірку. У k-кратній перехресній перевірці набір даних розбивається на k рівних підмножин. Потім запускається алгоритм ML, і оцінка виконується k разів, використовуючи одну підмножину як тестовий набір одночасно.

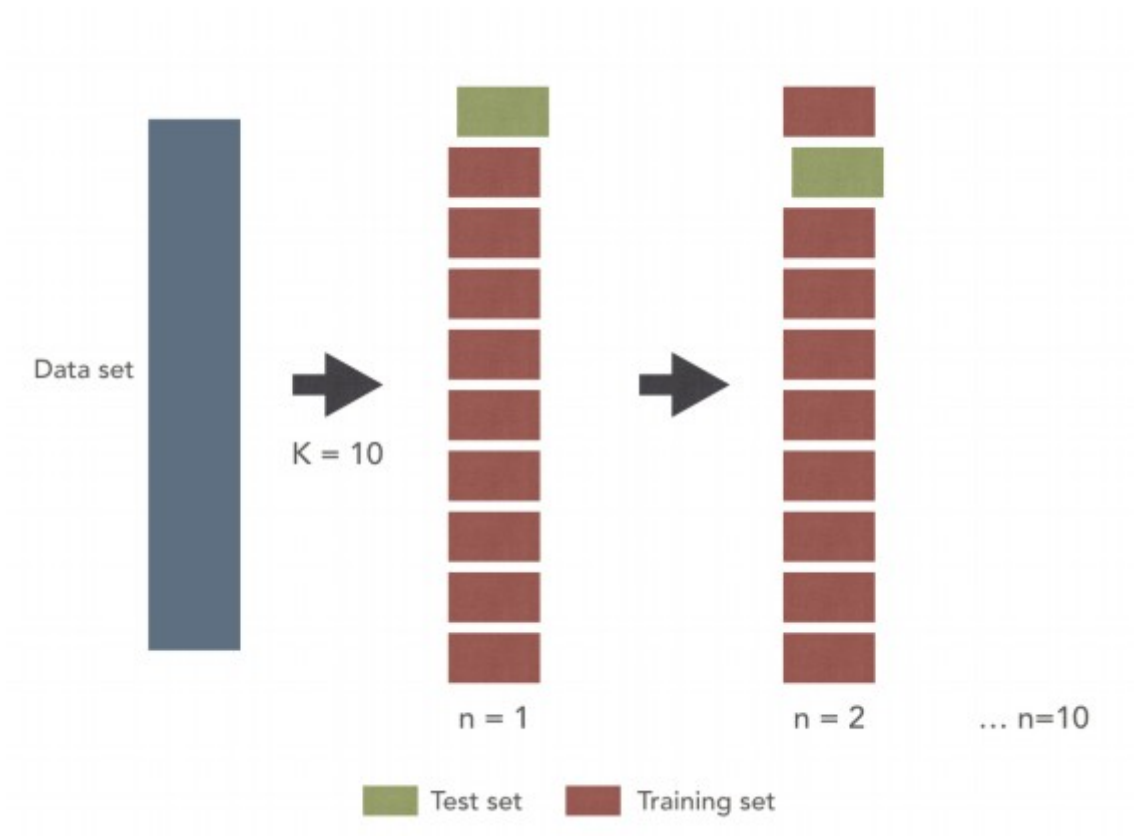


Рис. 2.1. Приклад поділу тестового набору та навчального набору в k-кратній перехресній перевірці

Оцінка точності, згідно з [7], розраховується як середнє значення точності k окремих показників точності. Рассел стверджує, що 5 і 10 є популярними значеннями k. Автор згадує значення k = 10 як загальноприйняту практику в застосунках прогнозного аналізу даних. В [12] використано 10-кратну перехресну перевірку в експериментальному дослідженні розпізнавання зображень. Рисунок 2.1 показує візуалізацію розбиття набору даних при 10-кратній перехресній перевірці.

Згідно з [13], результати на тестовому наборі можна відобразити у вигляді матриці невідповідностей з рядком та стовпцем для кожного можливого результату. Приклад матриці невідповідностей показано на рисунку 2.2. Кожен елемент матриці представляє кількість тестових прикладів, де фактичний результат - це рядок, а прогнозований результат - стовпець. Зелена діагональ представляє прогнози, які корелюють з фактичним результатом.

		Predicted outcome		
		A	**B**	**C**
Actual outcome	**A**	88	10	2
	B	14	40	6
	C	27	20	12

Рис. 2.2. Приклад матриці невідповідності

2.2. Бібліотеки машинного навчання

2.2.1. Проект scikit-learn

Для роботи з алгоритмами машинного навчання можна використовувати існуючі бібліотеки, призначені для цього завдання. Одна з бібліотек, яка підходить для цього проекту - scikit-learn. Проект scikit-learn був розпочатий у 2007 році [14]. З того часу розробка стала керованою спільнотою, і перший публічний реліз відбувся у 2010 році. В [15] стверджують, що scikit-learn тісно інтегрований з мовою програмування

Python, однією з найпопулярніших мов для машинного навчання. Також зазначають, що scikit-learn добре працює щодо часу обчислення порівняно з іншими обраними інструментами машинного навчання, доступними в Python. Бібліотека надає реалізації багатьох відомих алгоритмів машинного навчання. Вона підтримує, серед іншого, різні види алгоритмів регресії [16]. Бібліотека scikit-learn також має певну підтримку для штучних нейронних мереж (ANN), проте на своєму веб-сайті [17] рекомендує інші фреймворки для цього алгоритму. Крім того, scikit-learn надає функції для виконання k-кратної перехресної перевірки та створення матриць невідповідностей для перехресно перевірених даних [18].

Scikit-learn (також відома як sklearn) - це одна з найпопулярніших бібліотек машинного навчання з відкритим кодом для мови програмування Python. Вона надає широкий спектр інструментів для вирішення різноманітних задач машинного навчання, включаючи:

Основні Scikit-learn функції:

- Класифікація;
- Регресія;
- Кластеризація;
- Зменшення розмірності;
- Вибір ознак;
- Попередня обробка даних:
- Оцінка моделей.

Переваги полягають в наступному:

- Scikit-learn має простий та інтуїтивно зрозумілий API, що робить його легким у вивченні та використанні.

- Бібліотека надає велику кількість алгоритмів машинного навчання, що дозволяє вирішувати різноманітні задачі.

- Scikit-learn оптимізована для продуктивності та може обробляти великі набори даних і має детальну документацію з прикладами коду, що полегшує вивчення та використання бібліотеки.

- Scikit-learn є проектом з відкритим кодом, що означає, що він безкоштовний у використанні та розповсюдженні.

Scikit-learn - це потужна та універсальна бібліотека машинного навчання, яка є цінним інструментом для будь-якого фахівця з обробки даних або машинного навчання.

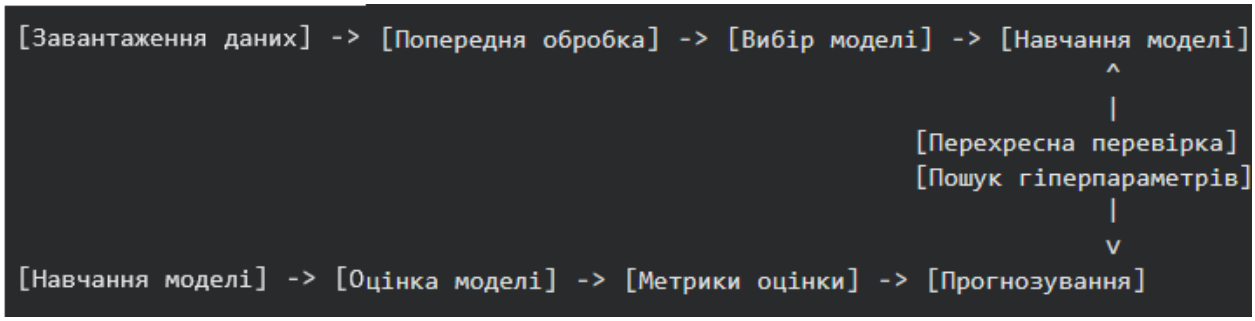


Рис. 2.3. Графічний алгоритм, який ілюструє основні компоненти та функціональність бібліотеки scikit-learn

Пояснення до алгоритму (рис. 2.3):

1. Завантаження даних: Спочатку дані завантажуються з різних джерел.

2. Попередня обробка: Дані готуються до навчання моделі, наприклад, шляхом масштабування, кодування категоріальних ознак тощо.

3. Вибір моделі: Вибирається відповідна модель машинного навчання, наприклад, лінійна регресія, дерево рішень, SVM тощо.

4. Навчання моделі: Модель навчається на підготовлених даних.

- Перехресна перевірка: Використовується для оцінки продуктивності моделі та вибору найкращих гіперпараметрів.

- Пошук гіперпараметрів: Використовується для оптимізації параметрів моделі.

5. Оцінка моделі: Оцінюється продуктивність навченої моделі за допомогою різних метрик.

- Метрики оцінки: Використовуються для вимірювання точності, повноти, F1-міри тощо.

6. Прогнозування: Навчена модель використовується для прогнозування на нових даних

2.2.2. Бібліотека наукових обчислень NumPy

Педрегоса [15] зазначає, що NumPy використовується зі scikit-learn для забезпечення базової структури даних для даних та параметрів моделі. NumPy - це бібліотека Python, яка надає інструменти для наукових обчислень та багатовимірних даних [19].

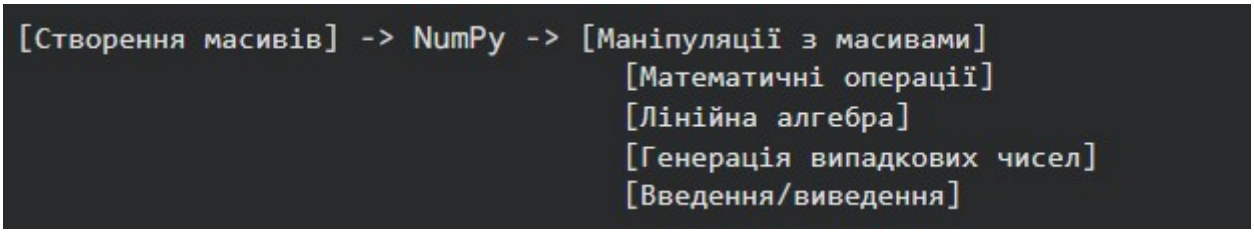


Рис. 2.4. Текстове представлення графічного алгоритму NumPy

Пояснення алгоритму показаного на рис.2.4:

1. Створення масивів: NumPy дозволяє створювати різні типи масивів, включаючи одновимірні, багатовимірні та структуровані масиви.

2. Маніпуляції з масивами: NumPy надає широкий спектр функцій для маніпуляцій з масивами, таких як зміна форми, об'єднання, розділення, індексація та зрізи.

3. Математичні операції: NumPy підтримує велику кількість математичних операцій над масивами, включаючи арифметичні операції, тригонометричні функції, експоненційні та логарифмічні функції тощо.

3. Лінійна алгебра: NumPy містить функції для виконання операцій лінійної алгебри, таких як множення матриць, знаходження визначника, оберненої матриці та власних значень.

4. Генерація випадкових чисел: NumPy дозволяє генерувати випадкові числа з різних розподілів, таких як рівномірний, нормальний та біноміальний.

5. Введення/виведення: NumPy надає функції для зчитування та запису даних з файлів різних форматів, таких як текстові файли, CSV, бінарні файли.

Наведемо додаткові можливості NumPy:

1. NumPy добре інтегрується з іншими бібліотеками Python, такими як SciPy, Matplotlib та scikit-learn.

2. бібліотека NumPy реалізована на мові C, що забезпечує високу продуктивність обчислень.

3. NumPy дозволяє виконувати операції над цілими масивами без використання циклів, що значно прискорює обчислення.

Цей графічний алгоритм (рис. 2.4) ілюструє основні функції NumPy та показує, як вони взаємодіють між собою. NumPy є фундаментальною бібліотекою для наукових обчислень у Python та широко використовується в різних галузях, таких як машинне навчання, обробка даних та аналіз даних.

2.2.3. Бібліотека TensorFlow

Для прогнозування за допомогою ANN можна використовувати TensorFlow [20]. Ця бібліотека розроблена Google. Її ядро реалізовано на C++, але пріоритетною мовою клієнта є Python. Tensorflow використовується в ряді проектів всередині Google, але він є відкритим кодом і може використовуватися іншими. Існує обгортка Python для Tensorflow під назвою Keras, яка має на меті спростити реалізацію ANN, надаючи високорівневий інтерфейс до низькорівневої мови, що використовується Tensorflow [22].

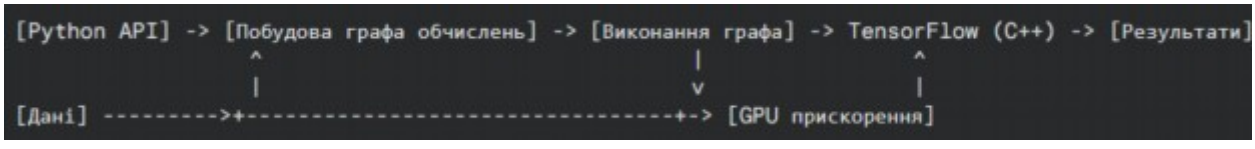


Рис. 2.5. Текстове представлення графічного алгоритму TensorFlow

Пояснення до рисунка 2.5:

- Python API: TensorFlow надає зручний Python API для визначення моделей машинного навчання.

- Побудова графа обчислень: TF будує граф обчислень, який представляє модель та її операції.

- Виконання графа: бібліотека виконує граф обчислень, використовуючи оптимізований C++ код.

- TensorFlow (C++): Ядро TensorFlow реалізовано на C++ для забезпечення високої продуктивності.

- GPU прискорення: TensorFlow може використовувати GPU для прискорення обчислень.

- Дані: Вхідні дані для моделі.

- Результати: Вихідні дані моделі.

Ця графічна інтерпретація ілюструє основні принципи роботи TensorFlow та показує, як різні компоненти взаємодіють між собою. TensorFlow є потужною бібліотекою для машинного навчання та глибокого навчання, яка використовується в багатьох галузях, таких як комп'ютерний зір, обробка природної мови та прогнозування.

Розглянемо основні концепції. Тензори – це базові структури даних у TensorFlow, які представляють багатовимірні масиви. Операції - функції, які виконуються над тензорами, наприклад, додавання, множення, згортка тощо. Граф обчислень - спрямований граф, який представляє обчислення в моделі. Вузли графа - це операції, а ребра - тензори, які передаються між операціями. Сесії - об'єкти, які керують виконанням графа обчислень.

2.3. Управління та зберігання інформації предметної області

Для зберігання інформації про лижі та тести лиж можна використовувати веб-сервіс разом з мобільним додатком. Таким чином, мобільні пристрої з встановленим додатком можна використовувати для доступу та редагування інформації незалежно від їх місцезнаходження, якщо вони підключені до Інтернету. Інформація, що зберігається в цьому веб-

сервісі, може згодом використовуватися алгоритмами машинного навчання для прогнозування найкращих лиж за заданих умов.

2.3.1. Веб-сервіс

Рішення для онлайн-сховища в цьому проекті - використання веб-фреймворку Django на основі Python. Django - це фреймворк, який був "розроблений для швидкого та легкого виконання завдань веб-розробки" [23]. Django надає функціональність для обробки HTTP-запитів і має вбудований менеджер об'єктно-реляційного відображення (ORM), здатний обробляти ряд систем баз даних на основі мови структурованих запитів (SQL). Моделі даних описуються в коді Python, і Django може надавати автоматизований інтерфейс адміністрування для бази даних. Однією з переваг використання Django для цього проекту є те, що Django працює на Python, що означає, що бібліотеку машинного навчання scikit-learn можна інтегрувати в нього.

Django - це високорівневий веб-фреймворк з відкритим кодом, написаний на Python, який слідує принципу "не повторюйся" (DRY). Він розроблений для того, щоб допомогти розробникам швидко створювати безпечні та масштабовані веб-застосунки.

Основні компоненти Django:

- Моделі. Django надає об'єктно-реляційне відображення (ORM), яке дозволяє розробникам взаємодіяти з базою даних за допомогою об'єктів Python, а не SQL-запитів.

- Шаблони. Django має потужну систему шаблонів, яка дозволяє розробникам розділяти логіку та представлення.

- Форми. Django спрощує створення та обробку форм, включаючи валідацію даних.

- Перегляди: Django надає функції для обробки запитів HTTP та генерації відповідей.

- URL-адресація: Django має гнучку систему URL-адресації, яка дозволяє розробникам створювати зрозумілі та зручні URL-адреси.

- Адміністративний інтерфейс: Django має вбудований адміністративний інтерфейс, який дозволяє легко керувати вмістом сайту.

Для передачі інформації з веб-фреймворку Django до інших веб-сервісів, веб-сайтів або мобільних додатків Django можна розширити за допомогою Django REST Framework [24]. Цей фреймворк надає, як випливає з назви, функціональність REST для Django. REST, по суті, за словами Баттла [25], являє собою набір операцій передачі стану: створення, читання, оновлення та видалення (CRUD). Ці операції в протоколі HTTP відображаються на POST, GET, PUT та DELETE. Django REST Framework також генерує веб-браузерний API для функціональності REST.

2.3.2. Мобільний додаток

Пристрої iOS є поширеними серед сервісерів у національній команді. Таким чином, було вирішено, що додаток буде розроблений для iOS.

Для розробки додатків для iOS від Apple можна використовувати мову Swift. Apple описує цю мову як "промислову мову програмування, яка є такою ж виразною та приємною, як мова сценаріїв" [14]. Swift є відкритим кодом, розробленим Apple та спільнотою відкритого коду.

Swift - це потужна та інтуїтивно зрозуміла мова програмування, створена компанією Apple для розробки додатків під iOS, macOS, watchOS та tvOS. Вона поєднує в собі сучасні підходи до програмування з багаторічним досвідом Apple у створенні програмного забезпечення.

Основні характеристики Swift:

- Swift втілює найновіші дослідження в області мов програмування, пропонуючи лаконічний синтаксис, іменовані параметри, типи, що виводяться, та багато інших сучасних функцій.

- Мова Swift розроблена з акцентом на безпеку, допомагаючи запобігати помилкам та підвищуючи надійність коду.

- Swift компілюється в машинний код, що забезпечує високу продуктивність програм.

- Мова Swift має простий та зрозумілий синтаксис, що робить його легким у вивченні та використанні, навіть для початківців.

- Swift Playground дозволяє експериментувати з кодом та бачити результати в реальному часі, що робить навчання цікавим та ефективним.

Swift швидший за Objective-C, попередню мову програмування Apple, має більш простий та зрозумілий синтаксис порівняно з Objective-C. Swift має багато сучасних функцій, які спрощують розробку та роблять код більш виразним.

2.4. Представлення методології розробки

Цей проект використовував веб-фреймворк Django та його розширення Django REST Framework для створення веб-сервісу, призначеного для обробки всієї інформації. Веб-фреймворк використовував базу даних для зберігання інформації. Програма (програма) для iOS була розроблена для забезпечення основного інтерфейсу кінцевого користувача. Зв’язок між веб-платформою та додатком реалізовано за допомогою викликів REST. Алгоритми машинного навчання були реалізовані для прогнозування ефективності лижних пар у заданих умовах.

Управління всією інформацією було реалізовано як веб-сервіс за допомогою Django разом із Django REST Framework. Мобільний додаток був розроблений як інтерфейс кінцевого користувача.

Для веб-сервісу було обрано Django. Цей вибір був зроблений тому, що він працює на Python, який спрощує запуск алгоритмів передбачення з використанням тієї самої мови. Усі сутності, такі як Test Session і Ski Pair, були закодовані як моделі даних Django, а менеджер Django ORM використовувався для перетворення моделей даних у таблиці в реляційній базі даних: SQLite. На рисунку 2.6 представлено найважливіші сутності проекту. Django REST Framework використовувався для реалізації серіалізаторів для всіх моделей, які надають JSON-представлення моделей.

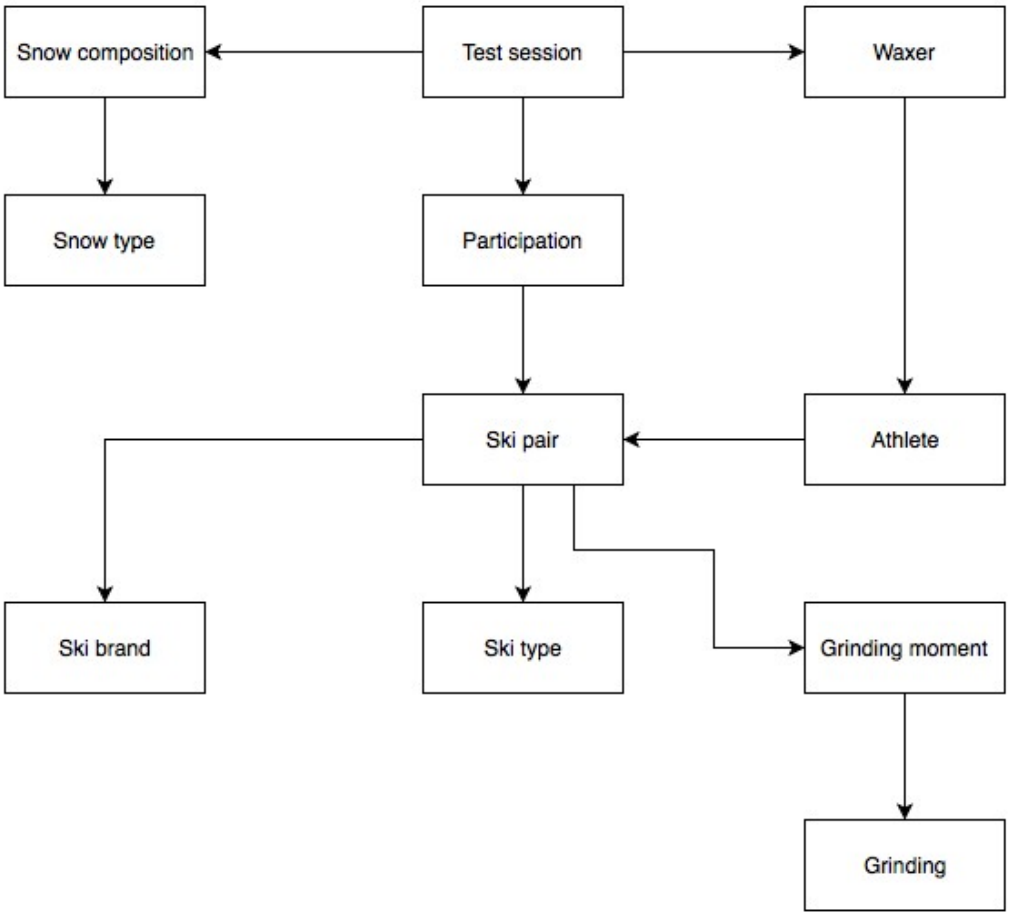


Рис. 2.6. Схема основних сутностей моделі даних

Функціональність CRUD була реалізована для більшості моделей, але деякі моделі були обмежені лише для читання. Ці моделі були сутностями, які вважалися дуже рідко оновлюваними або які не повинні оновлюватися іншими особами, крім адміністратора. Усі моделі, включно з функцією REST лише для читання, було додано до інтерфейсу автоматичного адміністрування Djangos, з якого їх усіх можна редагувати та видаляти. Інтерфейс адміністрування був обмежений керівником waxers.

2.5. Прогнозування продуктивності за допомогою машинного навчання

Щоб передбачити майбутню продуктивність лиж у заданих умовах, алгоритми ML були використані у серверній частині. Оскільки вхідні та вихідні дані можна отримати із зареєстрованих тестових сесій та лижних пар,

що беруть участь у них, використовувався тип навчання під наглядом. У цьому проекті прогнози були зроблені на лижах, упорядкованих за їх загальним враженням.

У проекті використовувалися бібліотеки Python scikit-learn та TensorFlow з Keras для роботи з алгоритмами ML. Для прогнозування використовувалися метод опорних векторів (SVM), штучні нейронні мережі (ANN) та дерева рішень. Для SVM було випробувано та оцінено розділення даних у просторі вищої розмірності.

2.5.1. Підготовка даних

Для підготовки вихідних даних для прогнозування, пари лиж, що брали участь у кожній тестовій сесії, були витягнуті з бази даних та розділені на одну з трьох категорій на основі рейтингу: Найкращий вибір, Ок та Відхилено. Кожна категорія мала представляти приблизно одну третину від загальної кількості лиж у тестовій сесії. Приклад того, як зробити поділ на категорії в тестовій сесії з 9 парами лиж, показано в таблиці 2.1.

Таблиця 2.1.

Приклад того, як пари лиж, що беруть участь у тестовій сесії, будуть класифіковані

Rank	Category
1	Top
2	Top
3	Top
4	Ok
5	Ok
6	Ok
7	Rejected
8	Rejected
9	Rejected

Призначена категорія кожної участі лиж була вихідною змінною в прогнозах. Поділ був мотивований пов'язаною роботою, описаною в цьому розділі, де результати були розділені на виграш, програш та нічию. Це також було схвалено національною командою, яка вважала це чітким способом інтерпретації результатів тестових сесій.

2.5.2. Налаштування алгоритмів машинного навчання

Оскільки визначення оптимальних параметрів для алгоритмів ML виходить за рамки цього проекту, в більшості випадків використовувалися стандартні параметри. Загалом, коли не вдавалося знайти стандартних налаштувань, вибір ґрунтувався на прикладах у посібниках користувача різних бібліотек, що використовувалися. Щоб вирішити, як налаштувати ANN, k-кратна перехресна перевірка запускалася кілька разів з використанням того самого навчального набору. В [4] стверджується, що не існує хорошої теорії для визначення структури та шарів ANN. Цю конфігурацію довелося визначати методом спроб і помилок. Перед обчисленням середньої точності ANN алгоритм був протестований з різною кількістю шарів та кількістю ітерацій зворотного поширення, щоб знайти конфігурацію з високою точністю. Конфігурація пробного раунду з найвищою точністю потім оцінювалася та використовувалася для порівняння з іншими алгоритмами ML, що використовуються в цьому проекті. Приклади з посібника користувача Keras використовувалися як відправна точка для параметрів.

Кількість вузлів у прихованих шарах у пробних раундах ґрунтувалася на емпіричних правилах, описаних в [26]. Одне з емпіричних правил полягало в тому, щоб визначити кількість нейронів, додавши 2/3 розміру вхідного шару до розміру вхідного шару. Це виявилося найкращим результатом з протестованих чисел. Автор також стверджує, що немає теоретичних причин використовувати більше 2 прихованих шарів. Тому було протестовано 1 та 2 шари. Під час випробувань було виявлено, що

оптимізатор під назвою Adam забезпечував кращу точність, ніж оптимізатор, запропонований у посібнику користувача Keras, і тому він використовувався для порівняння з іншими алгоритмами ML. Оптимізатор Adam описаний в [27].

2.5.3. Оцінка алгоритмів машинного навчання

Прогнози оцінювалися з використанням штучно згенерованих даних на основі реального досвіду, набутого національною командою. Дані були згенеровані для охоплення широкого спектру снігових умов.

Оцінка алгоритмів ML проводилася з використанням k-кратної перехресної перевірки. Для вимірювання продуктивності алгоритму ML використовувалася точність. Крім того, була створена матриця невідповідностей для візуалізації результатів кожного алгоритму ML. Результати алгоритмів ML порівнювалися з базовим рівнем, згенерованим алгоритмом ZeroR. Для забезпечення функціональності оцінювання використовувалася бібліотека scikit-learn.

2.5.4. Інструменти та архітектура

Прогнози були реалізовані разом з Django Framework та пов'язаними з ним пакетами. Pycharm використовувався як IDE. Використання пакетів Python Django Extensions та Werkzeug спростило налагодження веб-сервісу. Ці пакети дозволяють розробнику взаємодіяти зі сторінками помилок, представленими у веб-браузері. Таким чином, код можна було виконувати в його контексті для дослідження причин помилки. Venv використовувався для збереження всього коду python, пов'язаного з цим проектом, в ізольованому середовищі. Код python запускався на Python 3.6.

Додаток був розроблений для iOS, для чого потрібен комп'ютер Mac під управлінням macOS Sierra або вище. Xcode 9 був обраний як IDE. Оскільки Xcode розробляється та підтримується самою Apple, він вважався найбільш підтримуваним та стандартизованим середовищем для розробки

iOS. За допомогою Xcode додаток можна було встановити на пристрій iOS, фізично підключений до комп'ютера, а оператори друку в коді потім відображалися для налагодження в середовищі Xcode.

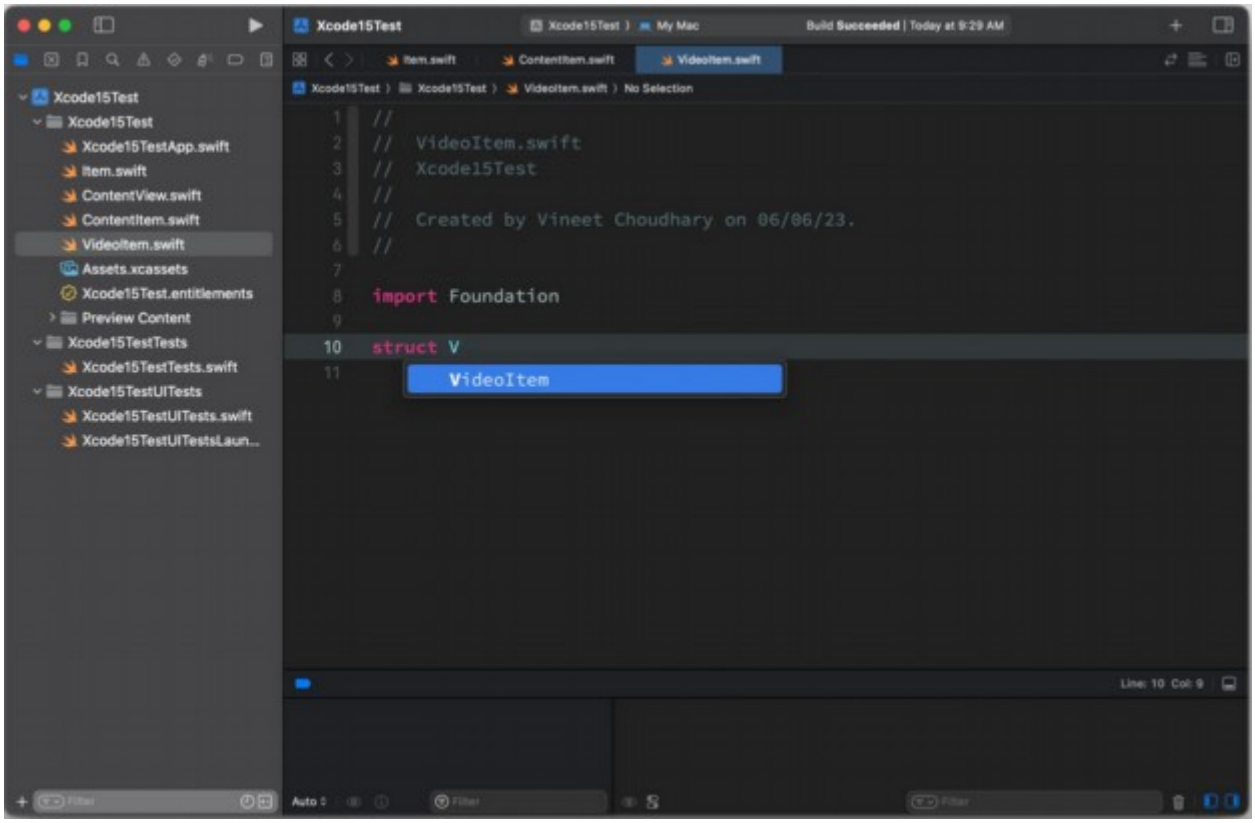


Рис. 2.7. Середовище Xcode

Для здійснення REST-викликів до веб-сервісу з програми використовувалася бібліотека Swift Alamofire разом з SwiftyJSON. AlamofireObjectMapper використовувався для перетворення відповідей з JSON на об'єкти Swift.

Для обробки залежностей бібліотеки для програми використовувався Cocoapods. Cocoapods працює шляхом визначення всіх залежностей у так званому podfile. Після цього можна запускати команди встановлення та оновлення через Cocapods для керування всіма залежностями.

Для керування історією коду та резервним копіюванням система контролю версій Git використовувалася як для веб-сервісу, так і для

програми. Робота постійно надсилалася до онлайн-репозиторіїв, розміщених на Bitbucket.

Під час проекту веб-сервіс та його база даних запускалися на локальному сервері розробки, наданому Django. Код машинного навчання був встановлений та запущений у тому ж середовищі. Додаток було встановлено на пристрої iOS шляхом їх безпосереднього підключення до комп'ютера розробки та встановлення через Xcode.

Висновки до розділу

В даному розділі досліджено методи, моделі та засоби, що забезпечують ефективне управління додатком на основі оцінок продуктивності, а також визначено ключові компоненти, які впливають на якість та ефективність роботи розроблених систем.

Методи оцінки продуктивності алгоритмів машинного навчання відіграють критичну роль у виборі оптимального підходу. Ці метрики забезпечують об'єктивну оцінку продуктивності алгоритмів у контексті предметної області. Бібліотеки машинного навчання, такі як scikit-learn та NumPy, були вивчені як основні інструменти для реалізації та оптимізації моделей. Scikit-learn надає широкий спектр алгоритмів машинного навчання, а також засоби для їх оцінки та налаштування. NumPy, як основа для обчислень, забезпечує ефективне управління багатовимірними даними, що є важливим для продуктивної роботи моделей.

Управління та зберігання інформації реалізовано через розробку веб-сервісу та мобільного додатка, що дозволяє інтегрувати зібрані дані та забезпечити доступ до них. Веб-сервіс слугує інтерфейсом для зберігання та обробки інформації, тоді як мобільний додаток спрощує взаємодію користувачів із системою.

Методологія розробки враховує всі аспекти від підготовки даних до побудови архітектури системи. Особливу увагу було приділено процесу

налаштування алгоритмів та забезпечення їх адаптивності до змінюваних умов.

Прогнозування продуктивності включає такі етапи:

- Підготовка даних, що охоплює очищення, нормалізацію та перетворення інформації для ефективної роботи моделей.
- Налаштування алгоритмів, яке забезпечує підбір оптимальних параметрів для досягнення точних результатів.
- Оцінка продуктивності, що визначає відповідність моделей практичним потребам через тестування на реальних даних.

Також представлено інструменти та архітектуру, які дозволяють інтегрувати алгоритми машинного навчання у додатки та забезпечити їх масштабованість. Запропоновані рішення гарантують гнучкість у використанні, високу продуктивність та доступність даних для користувачів.

Отже, цей розділ підсумував важливість використання сучасних методів машинного навчання та інтеграції їх з інструментами розробки додатків для досягнення високої ефективності та зручності у використанні.

РОЗДІЛ 3. ІМПЛЕМЕНТАЦІЯ МЕТОДІВ ТА АЛГОРИТМІВ УПРАВЛІННЯ ДОДАТКОМ НА ОСНОВІ ОЦІНОК ПРОДУКТИВНОСТІ

3.1. Етапи реалізації додатку для процесів прогнозування

Реалізація функціональності додатку була розділена на три етапи. Перший етап включав реалізацію веб-сервісу як REST API для отримання та зберігання даних. Другий етап полягав у створенні додатку та внесенні змін до REST API для відповідності потребам додатку. Третій етап полягав у реалізації алгоритмів машинного навчання для прогнозування продуктивності у веб-сервісі.

Першим кроком у реалізації веб-сервісу було визначення моделей Django, що представляють всі сутності в базі даних. Всі моделі були визначені в загальному файлі моделей. У цих моделях були визначені всі атрибути кожної сутності разом з їх обмеженнями. Запустивши команди python manage.py makemigrations і потім python manage.py migrate, моделі були перетворені та створені Django в базі даних SQLite.

Функціональність REST була побудована з використанням можливостей Django REST Framework, який надалі називатиметься DRF.

Маршрутизація URL була реалізована за допомогою вбудованої функціональності DRF. Кожна модель була зареєстрована для маршрутизації, що дозволило DRF обробляти запити CRUD та виконувати відповідні дії. Дії CRUD, за необхідності, були визначені та налаштовані у файлі набору переглядів. Додаток D містить реалізації коду з поясненнями для маршрутизації URL та наборів переглядів.

У кожному класі у файлі набору переглядів дозволи були обмежені, щоб вимкнути весь неаутентифікований доступ.

Були визначені серіалізатори для кожної моделі. Більшість логіки була оброблена DRF, але атрибути довелося вказати для кожної моделі.

За допомогою WritableNestedModelSerializer з додаткового пакета DRF Writable Nested16 поля, що містять зовнішні ключі до інших моделей, були реалізовані як вкладені серіалізатори, які можна було використовувати з усіма командами CRUD. Додаток Е містить приклад одного з серіалізаторів.

Побудова додатку спочатку була зосереджена на створенні класу для обробки всього зв'язку з веб-сервісом та створення класів, що відповідають сутностям в базі даних веб-сервісу. Другий етап полягав у реалізації користувацьких переглядів та контролерів переглядів.

Всі моделі у веб-сервісі також були реалізовані як класи в додатку. Відповідаючи загальному протоколу, ці класи можна було використовувати для запитів CRUD, зроблених через функції, пояснені в наступному розділі. У класах моделей використовувалося об'єктне відображення для перетворення відповідей JSON із запитів у відповідні класи.

Був написаний клас, для обробки всіх REST-викликів до веб-сервісу. Цей клас слідує шаблону Singleton, маючи статичну константу, що містить сам клас.

Функції зв'язку вищезгаданого класу використовують Alamofire для здійснення запитів до веб-сервісу. Відповіді на ці запити є асинхронними, що обробляється надсиланням сповіщень при отриманні відповіді або після тайм-ауту. Сповіщення, які слід використовувати для запитів, надаються у викликах до функцій зв'язку. Коли сповіщення спрацьовує, воно обробляється в контролері перегляду, який робить запит.

XCTest використовувався на ранніх етапах процесу побудови, щоб переконатися, що вся функціональність REST додатку працює правильно. Кожна операція CRUD була протестована, а їх асинхронні сповіщення очікувалися та перевірялися.

Контролери переглядів обробляють логіку для представлення переглядів та керують поведінкою користувацького інтерфейсу додатку.

Apple надає функцію для представлення вмісту в iOS-додатках під назвою UITableViews17. Вони широко використовувалися в додатку для

представлення різноманітного вмісту, як-от у контролері перегляду списку тестових сесій, де представлений список тестових сесій.

3.2. Прогнозування за допомогою машинного навчання

Прогнозування виконувалося у веб-сервісі, що означало прямий доступ до бази даних.

3.2.1. Підготовка даних

Як описано в другому розділі, формат вихідних даних прогнозів був побудований шляхом розділення участі лиж у кожній тестовій сесії на категорії. Межі рейтингу для кожної категорії обчислювалися шляхом множення кількості пар лиж, що беруть участь у кожній тестовій сесії, на 0,33 та 0,66 та округлення до найближчого цілого числа.

Алгоритми ML не можуть працювати з категоріальними даними без перетворення їх у числове представлення. У випадку цього проекту, основа лиж та шліфування кожної пари лиж, що брала участь, мали бути перетворені. Це було вирішено за допомогою OneHotEncoder з бібліотеки scikit-learn. OneHotEncoder створює бінарний стовпець для кожної категорії [29]. Наприклад, якщо основа лиж біла, а вибір - чорний та білий, стовпець для чорного містить нуль (0), а стовпець для білого містить одиницю (1).

Щоб уникнути того, що деякі атрибути даних матимуть більшу вагу, ніж інші в прогнозах, використовувалося масштабування ознак. Ця процедура стандартизації, за словами Боллегала [30], є важливим кроком у багатьох завданнях ML. Масштабування ознак здійснювалося за допомогою StandardScaler з бібліотеки scikit-learn [31]. Це призвело до того, що всі значення відповідали тим самим верхнім та нижнім межам.

3.2.2. Алгоритми

Прогнози робилися з використанням SVM, дерева рішень та ANN.

Для прогнозування за допомогою SVM класифікатор з scikit-learn був ініціалізований з ядром вищої розмірності під назвою "rbf". Це стандартне ядро алгоритму SVM18 у scikit-learn.

Як і для SVM, алгоритм дерева рішень був ініціалізований з класифікатора з scikit-learn.

Лістинг 3.1. Support Vector Machine

```
X, y = prepare_data()

# --Predicting with Kernel Support Vector Machine--
svm_classifier = SVC(kernel='rbf', random_state=0)

# Applying K-fold Cross Validation
accuracies = cross_val_score(estimator=svm_classifier,
                             X=X,
                             y=y.ravel(),
                             scoring='accuracy',
                             cv=10)

# Getting mean accuracy and standard deviation
mean_accuracy = accuracies.mean()
standard_deviation = accuracies.std()

# Creating confusion matrix by running predictions on the data k times
cross_val_predictions = cross_val_predict(svm_classifier,
                                          X,
                                          y.ravel(),
                                          cv=10)

cm = confusion_matrix(y, cross_val_predictions)
```

Щоб ANN могла обробляти вихідну змінну в навчальному наборі, її довелося перетворити на матрицю істинних та хибних значень. Однак вихідні дані алгоритму ANN були не чіткими категоріями, а ймовірностями для категорій. Тому було зроблено перетворення перед перетворенням результату назад в один стовпець. Це перетворення дало всім вихідним даним з ймовірністю більше 50% значення 1. Вихідні дані з меншою ймовірністю отримали значення 0.

Перед запуском алгоритму ANN його потрібно було налаштувати. Була вказана кількість вузлів у вхідному шарі. Також були налаштовані 2 приховані шари з певною кількістю вузлів та вихідний шар. Крім того, була вказана кількість епох, щоб повідомити алгоритму ANN, скільки разів ітерувати процес зворотного поширення. Зворотне поширення пояснюється в другому розділі.

Лістинг 3.2. ANN

```
def predict_using_decision_tree():
    X, y = prepare_data()

    # --Predicting with Decision Tree--
    decision_tree_classifier = DecisionTreeClassifier(criterion='entropy',
random_state=0)

    # Applying K-fold Cross Validation
    accuracies = cross_val_score(estimator=decision_tree_classifier,
                                 X=X,
                                 y=y.ravel(),
                                 scoring='accuracy',
                                 cv=10)

    # Getting mean accuracy and standard deviation
    mean_accuracy = accuracies.mean()
    standard_deviation = accuracies.std()

    # Creating confusion matrix by running predictions on the data k times
    cross_val_predictions = cross_val_predict(decision_tree_classifier,
                                              X,
                                              y.ravel(),
                                              cv=10)
    cm = confusion_matrix(y, cross_val_predictions)


def predict_using_ann():
    import tensorflow as tf
    import random as rn

    # The below is necessary for starting Numpy generated random numbers
    # in a well-defined initial state.

    np.random.seed(42)

    # The below is necessary for starting core Python generated random numbers
    # in a well-defined state.

    rn.seed(12345)

    # Force TensorFlow to use single thread.
    # Multiple threads are a potential source of non-reproducible results.
    # For further details, see: https://stackoverflow.com/questions/42022950/

    session_conf = tf.ConfigProto(intra_op_parallelism_threads=1,
                                  inter_op_parallelism_threads=1)

    from keras import backend as K

    tf.set_random_seed(1234)

    sess = tf.Session(graph=tf.get_default_graph(), config=session_conf)
    K.set_session(sess)

    X, y = prepare_data()
```

Більшість коду вище пояснено в коментарях. "input_dim" - це кількість вхідних параметрів, яких у цьому випадку 10. "units" - це кількість нейронів у шарі. Кількість одиниць у вихідному шарі відповідає стовпцям, створеним шляхом перетворення вихідної змінної. "epochs" - це кількість разів, коли процес зворотного поширення має повторитися.

Спочатку було виявлено, що показники точності відрізняються кожного разу, коли k-кратна перехресна перевірка запускається на алгоритмі ANN. Це було вирішено шляхом запуску коду, запропонованого на веб-сайті Keras [32], перед виконанням оцінки.

3.2.3. Оцінка точності алгоритмів

Для обчислення точності алгоритмів ML використовувалася функція з бібліотеки scikit-learn. Та ж бібліотека використовувалася для створення матриць невідповідностей для кожного алгоритму ML.

Як згадувалося в розділі методологія, базовий рівень був згенерований алгоритмом ZeroR. Це також оброблялося через бібліотеку scikit-learn. Оцінка докладніше описана в другому розділі.

Лістинг 3.3. ZeroR

```
X, y = prepare_data()

# --Predicting with ZeroR for comparison--
zeror_classifier = DummyClassifier(strategy='most_frequent')

# Applying K-fold Cross Validation
accuracies = cross_val_score(estimator=zeror_classifier,
                             X=X, y=y.ravel(),
                             scoring='accuracy',
                             cv=10)

# Getting mean accuracy and standard deviation
mean_accuracy = accuracies.mean()
standard_deviation = accuracies.std()

# Creating confusion matrix by running predictions on the data k times
cross_val_predictions = cross_val_predict(zeror_classifier,
                                          X, y.ravel(),
                                          cv=10)
cm = confusion_matrix(y, cross_val_predictions)
```

3.3. Результати застосування машинного навчання

Як показано на рисунку 3.1 цей проект дозволяє сервісерам використовувати додаток для зберігання результатів тестів лиж. Додаток надсилає дані до веб-сервісу, де вони зберігаються. Веб-сервіс також містить код для прогнозування за допомогою машинного навчання. Результати

представлені окремо у трьох розділах: Веб-сервіс, iOS-додаток та Прогнозування.

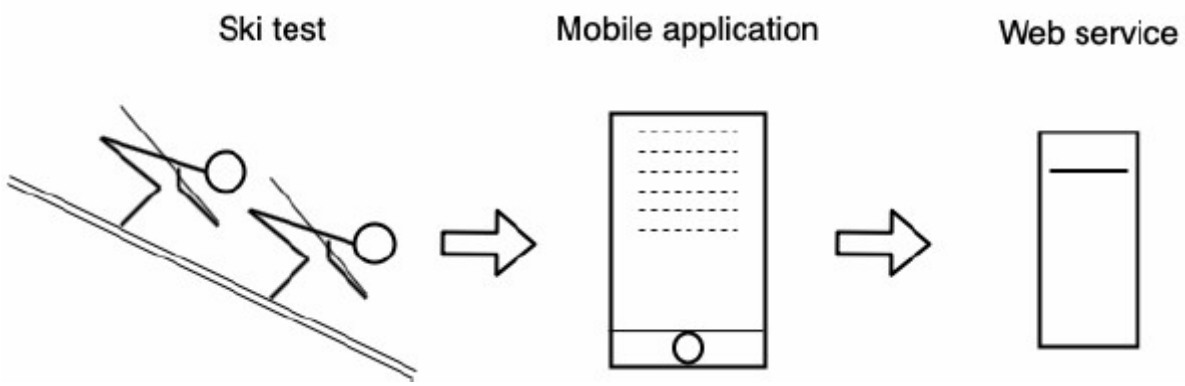


Рис. 3.1. Візуалізація робочого процесу тестування

Веб-сервіс був розроблений як основа проекту. Основним завданням веб-сервісу є обробка даних та забезпечення обмеженого доступу до них. Веб-сервіс зберігає всі дані, що використовуються додатком.

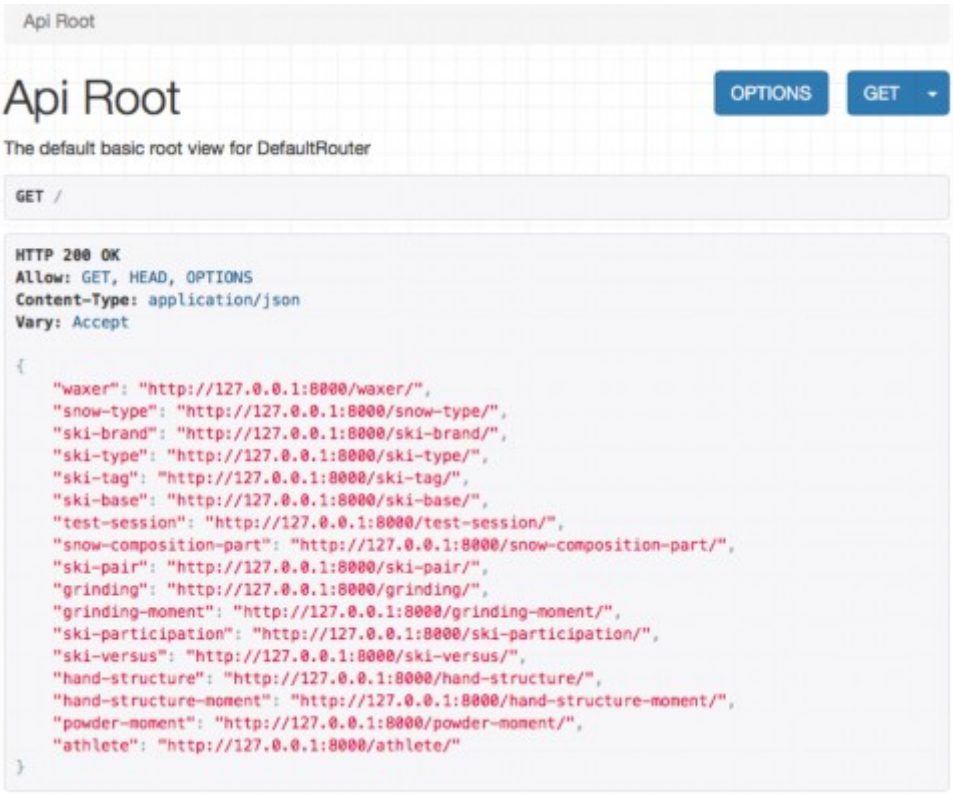


Рис. 3.2. Головна сторінка веб-перегляду REST API

Доступ до даних можна отримати через службу REST. Веб-сервіс також надає інтерфейс до REST API, доступний з веб-браузера, як показано на рисунку 3.2. Кожну сутність можна переглянути, а операції створення можна виконати з цього інтерфейсу.

З додатку користувачі можуть керувати парами лиж та тестовими сесіями разом з пов'язаними даними. Вони можуть додавати, редагувати та видаляти дані, а також ранжувати лижі, що беруть участь у кожній тестовій сесії. Під час запуску додатку користувачі зустрічаються з вікном входу. Ніхто не може пройти повз це вікно, якщо не надасть облікові дані користувача, які будуть схвалені веб-сервісом. Нижче представлений користувацький інтерфейс для обробки тестових сесій.

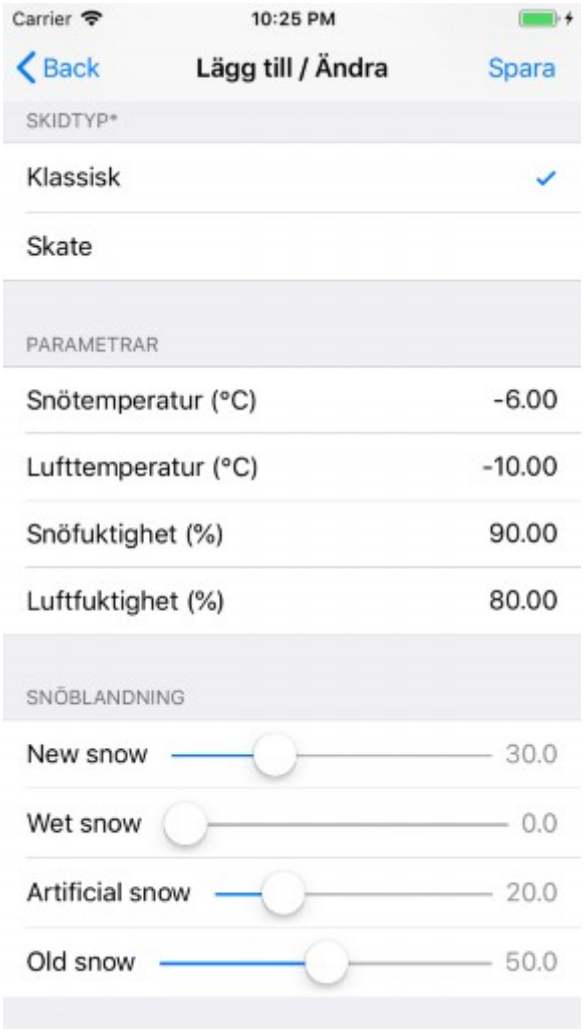


Рис. 3.3. Форма тестової сесії

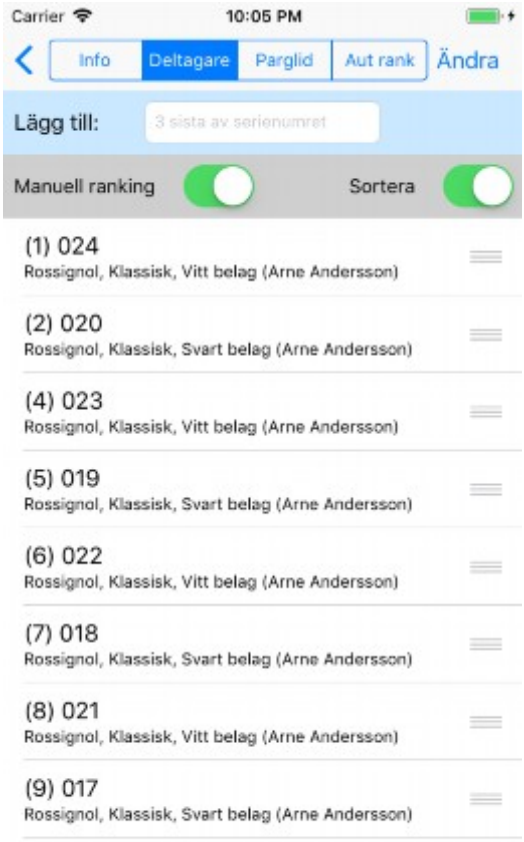


Рис. 3.4. Додавання та ручне ранжування

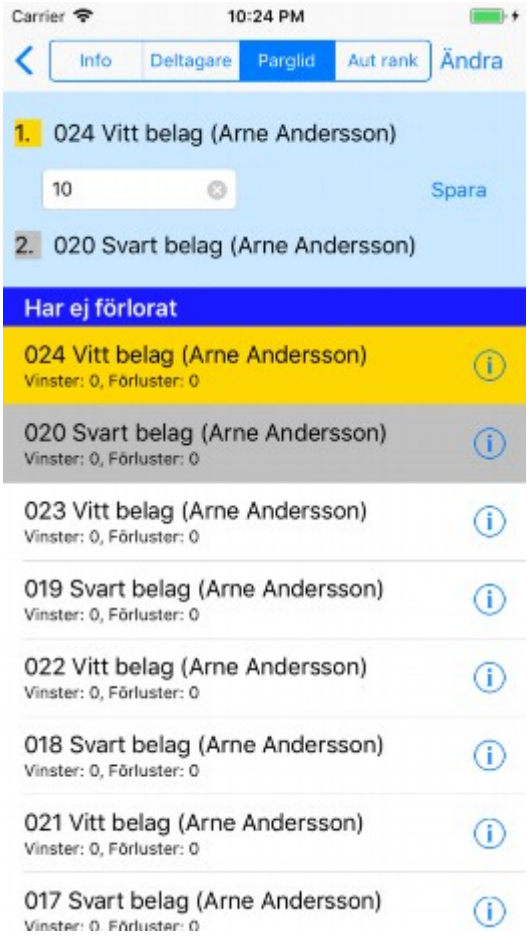


Рис. 3.5. Розрахований рейтинг

3.3.1. Обробка тестових сесій

Під час роботи з тестовими сесіями в додатку спочатку використовується форма, показана на рисунку 3.3, для створення тестової сесії. У цій формі вводяться снігові умови, тип лиж та інша інформація, пов'язана з самою сесією. Наступний крок - введення результатів тесту. Це можна зробити двома різними способами. Або шляхом ручного ранжування лиж, або шляхом введення результатів для серії паралельних ковзань та дозволу серверу розрахувати рейтинг. Другий спосіб не був повністю реалізований під час цього проекту.

3.3.2. Ручне ранжування

На рисунку 3.4 представлена сторінка, де можна додавати участі лиж до тестової сесії та виконувати ручне ранжування. Щоб додати участь лиж, користувач вводить перші 3 цифри серійного номера пари лиж. Потім додаток запитує веб-сервіс на відповідність пар лиж. Якщо є лише один збіг, пара лиж додається як участь. Якщо є більше одного збігу, ці пари лиж представлені у списку з деталями, і користувач може вибрати, яку пару лиж додати. Ручне ранжування можна активувати за допомогою перемикача "Ручне ранжування". Щоб змінити рейтинг лиж, потрібно активувати перемикач "Сортувати". Потім користувач може перетягувати кожну участь лиж, щоб змінити порядок ранжування.

3.3.3. Розрахований рейтинг

На рисунку 3.5 показана сторінка для ранжування лиж на основі тестів ковзання. Для кожного паралельного ковзання між двома парами лиж користувач вибирає пару лиж, що виграла, та пару лиж, що програла, і вводить відстань між парами лиж у текстовому полі. Якщо користувач повторює це, доки кожна пара лиж, крім однієї, не програє, користувачеві повідомляється, що можна виконати автоматичне ранжування. Оскільки сервісери не мали потреби в цій функції в даній ситуації, цей перегляд був

реалізований лише як концепція для майбутнього використання. Отже, функція автоматичного розрахунку рейтингу не була реалізована.

3.4. Прогнозування на основі оцінок продуктивності

Прогнози базуються на даних з бази даних веб-сервісу.

Таблиця 3.1.

Частина тестових даних, наданих алгоритмам ML

Test number	1	2	3	4	5
New snow	0%	100%	50%	0%	100%
Old snow	100%	0%	50%	100%	0%
Wet snow	0%	0%	0%	0%	0%
Air temperature	-15°C	-15°C	-15°C	-5°C	-5°C
Ranking	**Ski pair**	**Ski pair**	**Ski pair**	**Ski pair**	**Ski pair**
1	001	001	001	002	002
2	002	002	002	001	001
3	003	003	003	006	003
4	004	004	004	005	004
5	005	005	005	003	006
6	006	006	006	004	005
7	007	007	007	007	007
8	008	008	008	008	008

У таблиці 3.1 показано приблизно одну третину даних, створених для тестування алгоритмів ML. Дані також містять тестові сесії для відповідних снігових умов, коли температура повітря становить 0°C, 5°C та 10°C.

Далі в таблицях представлені дані про пари лиж та шліфування, що брали участь в оцінці алгоритмів машинного навчання. У таблиці 3.2 представлені снігові умови, за яких вони працюють найкраще.

Таблиця 3.2.

Представлені найкращі умови

Grindings	Best for condition
A	New or old snow, < -5°C
B	New or old snow < +2°C
C	New snow > 0°C
D	Wet old snow > 0°C

У таблиці 3.3 перераховані пари лиж разом з їх властивостями.

Таблиця 3.3.

Властивості об’єктів (лиж)

Serial Number	Ski type	Ski Base	Grinding
001	Cold Conditions	Black	A
002	Cold Conditions	Black	B
003	Cold Conditions	Black	C
004	Cold Conditions	Black	D
005	Cold Conditions	White	A
006	Cold Conditions	White	B
007	Cold Conditions	White	C
008	Cold Conditions	White	D
009	Universal	Black	A
010	Universal	Black	B
011	Universal	Black	C
012	Universal	Black	D
013	Universal	White	A
014	Universal	White	B
015	Universal	White	C
016	Universal	White	D
017	Klister	Black	A
018	Klister	Black	B
019	Klister	Black	C
020	Klister	Black	D
021	Klister	White	A
022	Klister	White	B
023	Klister	White	C
024	Klister	White	D

У таблиці 3.4 представлена середня точність та стандартне відхилення протестованих алгоритмів. Алгоритм ANN мав найвищу середню точність. Алгоритм дерева рішень з використанням ядра вищої розмірності був другим

найкращим, а алгоритм SVM був третім найкращим за точністю. Всі три алгоритми ML працювали краще, ніж базовий рівень ZeroR.

Таблиця 3.4.

Середня точність та стандартне відхилення для алгоритмів ML, що використовуються в проекті

Algorithm	Mean accuracy	Standard deviation
SVM	61.01%	15.94%
ANN	75.68%	12.20%
Decision Tree	67.75%	22.07%
ZeroR	38.36%	2.17%

Таблиця 3.5.

Матриці невідповідностей для прогнозів, зроблених у дослідженні

SVM Higher dimensional kernel			
	Top pick	Ok	Rejected
Top pick	13	14	1
Ok	6	20	15
Rejected	0	7	36

ANN			
	Top pick	Ok	Rejected
Top pick	21	5	2
Ok	3	27	11
Rejected	0	6	37

Decision Tree			
	Top pick	Ok	Rejected
Top pick	19	8	1
Ok	8	25	8
Rejected	2	8	33

ZeroR baseline			
	Top pick	Ok	Rejected
Top pick	0	0	28
Ok	0	0	41
Rejected	0	0	43

У таблиці 3.5 показано матрицю невідповідностей для алгоритмів ML та для базового алгоритму ZeroR. Вертикальні категорії представляють

фактичний результат тестового набору. Горизонтальні категорії представляють прогнозований результат.

3.5. Аналіз отриманих результатів

3.5.1. Особливості представлення інформації

У своєму поточному вигляді, списки в додатку отримують та представляють всі пов'язані дані з веб-сервісу кожного разу, коли перегляд відкривається або оновлюється. Зі зростанням бази даних це вплине на продуктивність та зручність використання додатку. Все більше даних доведеться отримувати з веб-сервісу, що збільшить час завантаження, і з більшою кількістю даних у списку користувачеві буде важче знайти те, що він шукає. Крім того, великі завантаження можуть бути дорогими з точки зору грошей. Надаючи користувачеві можливість фільтрувати вміст перед отриманням його з бази даних, можна зменшити розмір даних. Іншим способом обмеження даних, отриманих із сервера, є використання пагінації, де лише частина даних розбивається на кілька отримань. Більше даних можна було б отримати, коли користувач прокручує списки вниз.

Продовжуючи тему споживання даних, додаток був побудований таким чином, щоб оновлювати всі списки щоразу, коли користувач входить у перегляд. Мотивацією для цього було уникнення ситуацій, коли користувач намагається додати або видалити об'єкт, а ця дія вже була виконана іншим користувачем. Щоб зменшити трафік даних, відповідальність за оновлення списків можна було б перенести на користувача. UITableViews надає функціональність прокручування перегляду вгору за перший рядок для оновлення. Ця функціональність вже реалізована у всіх списках додатку.

3.5.2. Прогнозування

Під час оцінки прогнозів точність сильно варіювалася між кожним раундом у k-кратній перехресній перевірці. Алгоритми ML мали стандартне

відхилення приблизно від 12% до 22%. Однією з причин варіації, ймовірно, є те, що існують параметри, що впливають на результат, які невідомі.

На початку цього проекту було задумано використовувати дані з тестів лиж, проведених сервісерами. Після вивчення цих даних виявилося, що вони не містять достатньо інформації для використання в оцінці алгоритмів машинного навчання. Тому були створені штучні тестові дані. Як згадувалося в другому розділі, ці тестові дані були створені на основі відображення результатів реальних тестів. Однак одна відмінність від реального світу полягає в тому, що не було створено зразків, які б відхилялися від шаблону. Це, ймовірно, трапляється в реальному світі.

Один приклад з реального світу, коли пара лиж відхилилася від очікувань, походить з досвіду. Під час змагань одна пара лиж надзвичайно добре показала себе в умовах, де очікувалося, що вона буде погано працювати. Використовуючи категоризацію, представлену в цьому звіті, передбачалося, що вона буде в категорії відхилених, як сервісменом, який представляє виробника лиж, так і сервісерами. Можна стверджувати, що умови відрізнялися від умов на звичайних змагальних майданчиках поєднанням великої висоти, що могло вплинути на вологість повітря, та теплого сонця через південну широту. Цей досвід важливо пам'ятати під час тестування лиж у незвичайних умовах. Однак під час створення тестових даних стверджується, що результат більшості тестів лиж відповідає їхнім очікуванням. Підсумовуючи, можна сказати, що штучно створені тестові дані, використані в цьому проекті, були достатньо точними для використання в оцінці алгоритмів ML.

Визначення типу снігу та оцінка лиж здійснюються людьми, і це оцінки, які, ймовірно, будуть відрізнятися між спостерігачами і навіть між різними оцінками того самого спостерігача. Можна стверджувати, що оскільки середовище не повністю відоме та певне, прогнози, засновані на реальних даних, не можуть досягти 100% точності незалежно від обраного алгоритму.

Однією з цілей цього проекту було допомогти сервісерам у виборі найкращих лиж для заданих умов. Результат оцінки алгоритмів ML, використаних у цьому проекті, свідчить про те, що це можливо.

Навіть для невеликого набору даних, використаного в оцінці, середня точність для алгоритму ANN була аж 75,68% зі стандартним відхиленням 12,20%, і точність, ймовірно, зросте з більшою кількістю даних для навчання. Ймовірно, важливішим є те, що лише кілька зразків, 2 для алгоритму ANN та 1 для інших, були неправильно класифіковані як відхилені, коли їх фактична категорія була найкращим вибором. Це свідчить про те, що ризик того, що фактично найкраща пара лиж не буде обрана для тесту, є низьким.

Алгоритм ANN правильно передбачив більше всіх трьох категорій: найкращий вибір, ок та відхилено, ніж алгоритм дерева рішень та алгоритм SVM. K-кратна перехресна перевірка алгоритмів показала, що ANN має менше стандартне відхилення для значень точності. На закінчення, ANN здається найкраще підходить для даної проблеми, і він показав багатообіцяючі результати для майбутнього використання в реальному середовищі. Важливо зазначити, що підготовка до алгоритму ANN у цьому проекті була більш трудомісткою, ніж для інших алгоритмів. Це означає, що порівняння не є повністю справедливим. Якби на інші алгоритми було витрачено стільки ж часу на налаштування їх параметрів, вони могли б працювати краще. Однак, виходячи з досвіду автора цього проекту, малоймовірно, що їх продуктивність була б такою ж високою, як у ANN.

Є деякі можливі подальші розробки, які можна було б досить легко реалізувати в майбутньому розширенні цього проекту. Як згадувалося в цьому розділі, що вихідні дані алгоритму ANN - це ймовірності для кожної категорії. Замість перетворення вихідних даних на окремі категорії, це можна було б використовувати для створення прогнозованого списку рейтингів з парою лиж, яка, найімовірніше, буде найкращим вибором, ранжованою як номер один. Під час цього проекту не вистачило часу, щоб інтегрувати прогнози в додаток. Це можна було б зробити, дозволивши користувачеві

вводити снігові умови та вибирати пари лиж спортсменів. Ці параметри потім були б надіслані до веб-сервісу, який би використовував алгоритм ML для прогнозування найкращих лиж для умов, вказаних користувачем, та надсилав результат до додатку для представлення. Тестові дані, представлені в цьому звіті, були згенеровані на основі реального.

Висновки до розділу

Третій розділ зосереджено на імплементації методів та алгоритмів управління додатком, який ґрунтується на оцінках продуктивності. У межах цього розділу розглянуто етапи реалізації, застосування машинного навчання для прогнозування, результати роботи алгоритмів, а також аналіз отриманих даних.

Етапи реалізації додатку охоплювали проектування, розробку та інтеграцію компонентів, необхідних для процесів прогнозування. Було реалізовано архітектуру додатка, яка дозволяє ефективно працювати з даними, забезпечуючи гнучкість і масштабованість.

Прогнозування за допомогою машинного навчання включало підготовку даних та вибір відповідних алгоритмів. Підготовка охоплювала очистку, нормалізацію та трансформацію даних, що забезпечило високу якість вхідної інформації. Для прогнозування використовувалися різні алгоритми, які демонстрували свої переваги у специфічних умовах, таких як аналіз складних залежностей або робота з великими обсягами даних.

Результати застосування алгоритмів машинного навчання виявили їхню ефективність у реальних умовах. Серед основних досягнень:

- Обробка тестових сесій дозволила виявити сильні сторони обраних методів та вдосконалити їхню адаптацію до змінюваних умов.

- Ручне ранжування продемонструвало можливість інтеграції експертних оцінок у загальний процес прогнозування.

- Розрахований рейтинг надав кількісну оцінку результатів роботи алгоритмів, що спрощує їх аналіз.

Прогнозування на основі оцінок продуктивності дозволило отримати високоточні передбачення, які можуть бути застосовані для підтримки прийняття рішень у різних сценаріях. Використання оцінок продуктивності допомогло адаптувати моделі до специфіки задачі та забезпечити їхню релевантність.

Аналіз отриманих результатів підтвердив ефективність обраних підходів та алгоритмів. Особливу увагу приділено представленню інформації у зрозумілому для користувачів форматі, що сприяє її інтерпретації. Прогнозування, здійснене на основі розроблених методів, виявилося точним і надійним, що свідчить про високу якість реалізованої системи.

Таким чином, у цьому розділі вдалося продемонструвати практичну реалізацію теоретичних розробок та досягнути значних результатів у побудові прогнозної системи, яка забезпечує ефективне управління додатком на основі оцінок продуктивності.

ВИСНОВКИ

У магістерській роботі було здійснено комплексне дослідження проблеми прогнозування в заданій предметній області, розроблено методи, моделі та засоби управління додатком на основі оцінок продуктивності, а також реалізовано практичну імплементацію запропонованих підходів.

У першому розділі проведено ґрунтовний аналіз існуючих підходів до прогнозування. Визначено ключові проблеми, пов’язані з високою динамічністю даних, складністю побудови моделей та необхідністю врахування багатофакторності процесів. Розглянуто вплив основних факторів, таких як якість вхідних даних, обчислювальні ресурси та параметри алгоритмів, на продуктивність системи. Проведений аналіз алгоритмів машинного навчання (штучних нейронних мереж, методу опорних векторів, дерева рішень та базового алгоритму ZeroR) дозволив виділити їхні переваги та обмеження залежно від конкретних умов використання.

У другому розділі розроблено методологію управління додатком на основі оцінок продуктивності. Розглянуто методи оцінки продуктивності алгоритмів, включаючи використання відповідних метрик, таких як точність, середньоквадратична помилка та стабільність результатів. Використано сучасні бібліотеки машинного навчання, зокрема scikit-learn і NumPy, які забезпечують ефективну реалізацію алгоритмів та обробку багатовимірних даних. Важливу роль відведено організації управління та зберігання даних, реалізованих через веб-сервіс та мобільний додаток, які забезпечують зручний доступ користувачів до системи. Розроблена методологія включає підготовку даних, налаштування моделей та їх оцінку, що дозволяє інтегрувати алгоритми машинного навчання у практичні сценарії.

У третьому розділі описано процес імплементації методів та алгоритмів у систему управління додатком. Реалізовано етапи розробки додатка для прогнозування, які включають очищення та трансформацію даних, вибір і

налаштування алгоритмів, а також тестування їх продуктивності. Результати застосування алгоритмів машинного навчання, таких як обробка тестових сесій, ручне ранжування та розрахунок рейтингу, продемонстрували їх ефективність і здатність адаптуватися до змінюваних умов. Аналіз отриманих результатів підтвердив точність прогнозування та відповідність запропонованих рішень поставленим задачам.

Основні наукові та практичні результати роботи:

1. Проведено аналіз сучасних методів і алгоритмів прогнозування та їх адаптації до заданої предметної області.

2. Розроблено методологію управління додатком, яка базується на оцінках продуктивності алгоритмів машинного навчання.

3. Реалізовано інтегровану систему, що включає веб-сервіс та мобільний додаток, для управління даними та виконання прогнозування.

4. Проведено оцінку ефективності алгоритмів у реальних умовах, що дозволило оптимізувати процес прийняття рішень.

Отримані результати мають теоретичну та практичну цінність, оскільки дозволяють розробляти прогностичні системи, які враховують специфіку предметної області та забезпечують високу точність прогнозів. Розроблені підходи можуть бути використані для вирішення аналогічних задач у суміжних галузях.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. R. P. Bunker and F. Thabtah, “A machine learning framework for sport result prediction,” Appl. Comput. Inform., Sep. 2017.
2. R. P. Bonidia, L. A. L. Rodrigues, A. P. Avila-Santos, D. S. Sanches, and J. D. Brancher, “Computational Intelligence in Sports: A Systematic Literature Review,” Advances in Human-Computer Interaction, 2018. [Online]. Available: https://www.hindawi.com/journals/ahci/2018/3426178/.
3. N. Pathak and H. Wadhwa, “Applications of Modern Classification Techniques to Predict the Outcome of ODI Cricket,” Procedia Comput. Sci., vol. 87, pp. 55–60, 2016.
4. K. H. Brodersen, C. S. Ong, K. E. Stephan, and J. M. Buhmann, “The Balanced Accuracy and Its Posterior Distribution,” 2010, pp. 3121–3124.
5. X. Glorot, A. Bordes, and Y. Bengio, “Deep Sparse Rectifier Neural Networks,” p. 9.
6. I. H. Witten and E. Frank, Data mining: practical machine learning tools and techniques, 2nd ed. Amsterdam ; Boston, MA: Morgan Kaufman, 2005.
7. “The Swift Programming Language (Swift 4.1): About Swift.” [Online]. Available: https://developer.apple.com/library/content/documentation/Swift/Conceptual/Swift_Programming_Language/index.html#//apple_ref/doc/uid/TP40014097-CH3-ID0.
8. F. Pedregosa et al., “scikit-learn: Machine Learning in Python,” J. Mach. Learn. Res., vol. 12, p. 2825−2830, Oct. 2011.
9. “API Reference — scikit-learn 0.19.1 documentation.” [Online]. Available: http://scikitlearn.org/stable/modules/classes.html#module-sklearn.linear_model.
10. “3.1. Cross-validation: evaluating estimator performance — scikit-learn 0.19.2documentation.” [Online]. Available: https://scikit-learn.org/0.19/modules/cross_validation.html.
11. “NumPy — NumPy.” [Online]. Available: http://www.numpy.org/.

12. “Neural Network,” TensorFlow. [Online]. Available: https://www.tensorflow.org/api_guides/python/nn
13. Heger, D. (2008). "Evaluating the impact of Java enterprise edition application server tuning." ACM SIGMETRICS Performance Evaluation Review, 36(3), 10-20.
14. Menasce, D. A. et al. (2004). "Performance by design: computer capacity planning by example." Prentice Hall.
15. Feng, S. et al. (2015). "Real-time application performance management in cloud computing environments." Journal of Parallel and Distributed Computing, 75, 101-116.
16. Zhang, L. et al. (2017). "Dynamic resource allocation for fault-tolerant management in cloud applications." Future Generation Computer Systems, 79, 34-46.
17. Lu, Z. et al. (2020). "Scalable performance modeling for big data applications in distributed systems." ACM Transactions on Modeling and Computer Simulation, 30(1), 1-27.
18. Kleppmann, M. (2017). Designing Data-Intensive Applications. O'Reilly Media.
19. Menasce, D. A., & Almeida, V. A. F. (2002). Capacity Planning for Web Services: Metrics, Models, and Methods. Prentice Hall.
20. Jain, R. (1991). The Art of Computer Systems Performance Analysis. Wiley.
21. R. Budde and A. Himes, “High-resolution friction measurements of cross-country ski bases on snow,” Sports Eng., vol. 20, no. 4, pp. 299–311, Dec. 2017.
22. M. Swarén, “Experimental test setups and simulations in skiing mechanics,” Engineering Sciences, Royal Institute of Technology, Stockholm, 2014.
23. M. Bäckström, L. Dahlen, and M. Tinnsten, “Essential ski characteristics for crosscountry skis performance,” p. 6, 2008.

24. S. J. Russell and P. Norvig, Artificial intelligence: a modern approach, Third [revised] edition, Global edition. Boston Columbus Indianapolis New York San Francisco: Pearson, 2016.
25. Zhu, Y. et al. (2019). "Adaptive performance tuning for microservices in containerized environments." IEEE INFOCOM 2019 Conference.
26. Klein, C. et al. (2017). "Performance-driven application management in multi-cloud environments." Proceedings of ACM Symposium on Cloud Computing (SoCC).
27. Singh, A. et al. (2018). "Performance anomaly detection in cloud applications." Journal of Systems and Software, 139, 73-89.
28. Park, J. et al. (2016). "Application-centric resource management." IEEE Transactions on Cloud Computing, 4(1), 42-56.
29. InfluxData (2023). "Comparative review of APM tools: Splunk, AppDynamics, and Datadog." Source InfluxData
30. Liu, H. et al. (2021). "Machine learning for APM: Analyzing logs and metrics." International Journal of Cloud Computing, 9(2), 123-137.
31. Chen, X. et al. (2015). "Big data analytics in APM for web applications." IEEE Transactions on Big Data, 1(1), 47-57.
32. Osman, I. et al. (2022). "Real-time anomaly detection for distributed applications." Journal of Systems Architecture, 127, 102744.
33. Liu, X. et al. (2020). "Scalable APM techniques for edge computing applications." IEEE Internet of Things Journal, 7(4), 3430-3442.
34. Zhang, Z. et al. (2021). "Energy-aware application performance management in mobile systems." ACM Transactions on Embedded Computing Systems, 20(1), 1-19.
35. Cheng, L. et al. (2019). "Optimization techniques in APM: A survey." IEEE Access, 7, 72632-72647.
36. Bui, T. et al. (2023). "Hybrid methods for improving application throughput." Future Generation Computer Systems, 139, 79-95.

37. Kumar, R. et al. (2018). "Dynamic profiling for cloud-native application management." Proceedings of the ACM Symposium on Cloud Computing (SoCC).
38. D. Delen, D. Cogdell, and N. Kasap, “A comparative analysis of data mining methods in predicting NCAA bowl outcomes,” Int. J. Forecast., vol. 28, no. 2, pp. 543–552, Apr. 2012.
39. M. Gao, L. Yin, and J. Ning, “Artificial neural network model for ozone concentration estimation and Monte Carlo analysis,” Atmos. Environ., vol. 184, pp. 129–139, Jul. 2018.
40. T. Dietterich, “Overfitting and undercomputing in machine learning,” ACM Comput. Surv., vol. 27, no. 3, pp. 326–327, Sep. 1995.
41. Huang, J. et al. (2017). "Efficient performance monitoring for microservices architectures." International Conference on Distributed Computing Systems (ICDCS).
42. Gupta, S. et al. (2020). "Advanced metrics for monitoring in serverless environments." International Conference on Cloud Computing Technology and Science (CloudCom).
43. Bondi, A. B. (2014). Foundations of Software and System Performance Engineering. Addison-Wesley.
44. Barroso, L. A., & Hölzle, U. (2018). The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines. Morgan & Claypool.
45. Wang, Y. et al. (2021). "Predictive analytics for application scaling in cloud platforms." Journal of Cloud Computing, 10(1), 1-15.
46. Ding, C. et al. (2019). "Resource provisioning strategies in APM." ACM Computing Surveys, 52(3), 1-35.
47. Rao, J. et al. (2023). "AI-driven application tuning for performance enhancement." IEEE Transactions on Neural Networks and Learning Systems, 34(4), 2345-2360.

48. Klein, D. et al. (2019). "Distributed performance metrics in microservices." International Journal of Computer Applications, 12(3), 234-249.
49. Lo, D. et al. (2020). "Distributed tracing for performance bottleneck identification." IEEE Transactions on Cloud Computing, 8(3), 515-528.

ДОДАТКИ

Додаток А

Визначення моделей

Цей додаток показує вміст файлу моделей у веб-сервісі. HandStructure, HandStructureMoment та PowderMoment були створені для майбутнього використання та не включені до жодної функціональності, пов'язаної з цим проектом.

Лістинг А.1.

```
from django.db import models import django.contrib.auth
from django.contrib.auth.models import User
from django.db.models import Q
from django.db.models.signals import post_save
from django.dispatch import receiver
from rest_framework.exceptions import ValidationError


class Team(models.Model):

    """The team/organisation/sport"""
    name = models.CharField(max_length=30)

    def __str__(self):
        return self.name


class Profile(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    team = models.ForeignKey(Team, on_delete=models.CASCADE,
                             null=True)
    athletes = models.ManyToManyField('Athlete', blank=True)

    def __str__(self):
        return self.user.first_name + ' ' + self.user.last_name +
               '(' + self.user.username + ')'


# Defining signals so the Profile model will be automatically
created/updated when creating/updating User instances
@receiver(post_save, sender=User)
def create_user_profile(sender, instance, created, **kwargs):
    if created:
        Profile.objects.create(user=instance)


@receiver(post_save, sender=User)
def save_user_profile(sender, instance, **kwargs):
    instance.profile.save()


class SnowType(models.Model):
    team = models.ForeignKey(Team, on_delete=models.CASCADE)
    name = models.CharField(max_length=30)
    description = models.CharField(max_length=200, blank=True)

    def __str__(self):
        return self.name


class SkiBrand(models.Model):
    team = models.ForeignKey(Team, on_delete=models.CASCADE)
    name = models.CharField(max_length=30)
```

```
    # Overriding save to store team from request
    # TODO: Should be changed or removed in future use with
      multiple teams!
    def save(self, **kwargs):
        self.team_id = 1
        super(SkiBrand, self).save(**kwargs)

    def __str__(self):
        return self.name


class SkiType(models.Model):
    team = models.ForeignKey(Team, on_delete=models.CASCADE)
    name = models.CharField(max_length=30)
    description = models.CharField(max_length=200, blank=True)

    def __str__(self):
        return self.name


class SkiTag(models.Model):
    team = models.ForeignKey(Team, on_delete=models.CASCADE)
    name = models.CharField(max_length=30)
    description = models.CharField(max_length=200, blank=True)

    def __str__(self):
        return self.name


class SkiBase(models.Model):
    team = models.ForeignKey(Team, on_delete=models.CASCADE)
    name = models.CharField(max_length=30)
    description = models.CharField(max_length=200, blank=True)

    def __str__(self):
        return self.name


class TestSession(models.Model):
    team = models.ForeignKey(Team, on_delete=models.CASCADE)
    created_by =
      models.ForeignKey(django.contrib.auth.get_user_model(),
                        on_delete=models.PROTECT)
    time_of_test = models.DateTimeField()
    name_of_place = models.CharField(max_length=30)
    latitude = models.DecimalField(max_digits=13, decimal_places=10,
                                   null=True, blank=True)
    longitude = models.DecimalField(max_digits=13,
                                    decimal_places=10, null=True,
                                    blank=True)
    comment = models.CharField(max_length=300, blank=True)
    air_temperature = models.DecimalField(max_digits=4,
                      decimal_places=2, null=True, blank=True)
    snow_temperature = models.DecimalField(max_digits=4,
                       decimal_places=2, null=True, blank=True)
    air_humidity = models.DecimalField(max_digits=5,
                   decimal_places=2, null=True, blank=True)
    snow_humidity = models.DecimalField(max_digits=5,
                    decimal_places=2, null=True, blank=True)
    ski_type = models.ForeignKey(SkiType, on_delete=models.PROTECT)
    participating_skis = models.ManyToManyField("SkiPair",
                         through="SkiParticipation")
```

```
    def __str__(self):
        return self.time_of_test.__str__() + ' ' +
               self.name_of_place


class SnowCompositionPart(models.Model):
    test_session = models.ForeignKey(TestSession,
                   on_delete=models.CASCADE)
    snow_type = models.ForeignKey(SnowType,
                   on_delete=models.PROTECT)
    percentage = models.PositiveSmallIntegerField()

    def __str__(self):
        return self.snow_type.__str__() + ' ' + str(self.percentage)
               + '%' + self.test_session.__str__()


class SkiPair(models.Model):
    team = models.ForeignKey(Team, on_delete=models.CASCADE)
    owner = models.ForeignKey('Athlete', null=True,
                              on_delete=models.SET_NULL)
    serial_number = models.CharField(max_length=30)
    brand = models.ForeignKey(SkiBrand, on_delete=models.PROTECT)
    type = models.ForeignKey(SkiType, on_delete=models.PROTECT)
    tags = models.ManyToManyField(SkiTag, blank=True)
    nickname = models.CharField(max_length=30, blank=True)
    description = models.CharField(max_length=200, blank=True)
    base = models.ForeignKey(SkiBase, on_delete=models.PROTECT)
    weight_for_measure = models.PositiveSmallIntegerField(
                         verbose_name= 'Weight as base for the
                         measurements', null=True)
    height_full_body_weight = models.DecimalField(max_digits=4,
                              decimal_places=1, null=True,
                              blank=True)
    height_half_body_weight = models.DecimalField(max_digits=4,
                              decimal_places=1, null=True,
                              blank=True)
    weight_for_complete_pushdown = models.DecimalField(max_digits=5,
                                   decimal_places=2, null=True,
                                   blank=True)

    def __str__(self):
        if self.nickname:
            nickname_string = ' ' + self.nickname
        else:
            nickname_string = ''
        return self.serial_number[-3:] + nickname_string + ' ' +
               self.type.name


class Grinding(models.Model):
    team = models.ForeignKey(Team, on_delete=models.CASCADE)
    name = models.CharField(max_length=30)
    description = models.CharField(max_length=200, blank=True)

    def __str__(self):
        return self.name


class GrindingMoment(models.Model):
    date = models.DateTimeField()
    ski_pair = models.ForeignKey(SkiPair, on_delete=models.CASCADE)
```

```
    def __str__(self):
        return self.ski_pair.__str__() + ': ' +
               self.grinding.__str__()


class SkiParticipation(models.Model):
    ski_pair = models.ForeignKey(SkiPair, on_delete=models.CASCADE)
    test_session = models.ForeignKey(TestSession,
                                     on_delete=models.CASCADE)
    manual_rank = models.IntegerField(null=True, blank=True)
    calculated_rank = models.IntegerField(null=True, blank=True)
    comment = models.CharField(max_length=200, blank=True)

    # Makes sure related Ski Versuses are deleted if a
      participation is deleted
    def delete(self):
        related_versuses = SkiVersus.objects.filter(
            Q(test_session=self.test_session, winner=self.ski_pair)
              | Q(test_session=self.test_session,
                looser=self.ski_pair))
        if related_versuses:
            related_versuses.delete()
        super(SkiParticipation, self).delete()

    def __str__(self):
        return self.ski_pair.__str__() + ' ' +
               self.test_session.__str__() + ' Man. rank: ' + str(
               self.manual_rank) + ' Calc. rank: ' +
               str(self.calculated_rank)


class SkiVersus(models.Model):
    test_session = models.ForeignKey(TestSession,
                   on_delete=models.CASCADE)
                   winner = models.ForeignKey(SkiPair,
                   related_name='winning_pair',
                   on_delete=models.CASCADE)
    looser = models.ForeignKey(SkiPair, related_name='loosing_pair',
                               on_delete=models.CASCADE)
    difference = models.SmallIntegerField(verbose_name='Difference
                                           in centimeters')
    comment = models.CharField(max_length=200, blank=True)

    def validate_unique(self, exclude=None):
        qs = SkiVersus.objects.filter(
             test_session=self.test_session)
        if qs.filter(winner=self.winner,
                     looser=self.looser).exists():
            raise ValidationError('Ski Versus must have a unique
                                   combination of test session,
                                   winner and looser')

    def save(self, *args, **kwargs):
        self.validate_unique()

        super(SkiVersus, self).save(*args, **kwargs)

    def __str__(self):
        return "1:a " + str(self.winner.serial_number + ', 2:a ' +
               self.looser.serial_number)
```

```
    team = models.ForeignKey(Team, on_delete=models.CASCADE)
    name = models.CharField(max_length=30)
    description = models.CharField(max_length=200, blank=True)

    def __str__(self):
        return self.name


class HandStructureMoment(models.Model):
    date = models.DateTimeField()
    ski_pair = models.ForeignKey(SkiPair, on_delete=models.CASCADE)
    hand_structure = models.ForeignKey(HandStructure,
                     on_delete=models.CASCADE)

    def __str__(self):
        return self.ski_pair + ': ' + Grinding


class PowderMoment(models.Model):
    ski_pair = models.ForeignKey(SkiPair, on_delete=models.CASCADE)
    date = models.DateTimeField()

    def __str__(self):
        return 'Powdered ' + self.date


class Athlete(models.Model):
    team = models.ForeignKey(Team, on_delete=models.CASCADE)
    first_name = models.CharField(max_length=30)
    last_name = models.CharField(max_length=30)
    fis_code = models.IntegerField(null=True, blank=True,
                                   unique=True)
    ski_brand = models.ForeignKey(SkiBrand,
                                  on_delete=models.PROTECT)

    GENDER_CHOICES = (
        ('M', 'Male'),
        ('F', 'Female'),
    )
    gender = models.CharField(max_length=1, choices=GENDER_CHOICES)

    def __str__(self):
        return self.first_name + ' ' + self.last_name + ' ' +
               self.ski_brand.name
```