Одеський національний університет імені І. І. Мечникова
Факультет математики, фізики та інформаційних технологій
Кафедра оптимального керування та економічної кібернетики

Дипломна робота

бакалавра

на тему: **«Використання алгоритмів природних обчислень у машинному навчанні»**

«Application of nature-inspired algorithms in Machine Learning»

Виконала: студентка денної форми навчання
спеціальності 113 Прикладна математика
Литвиненко Інна Анатоліївна

Керівник: канд. фіз.-мат. наук, доцент Страхов Є.М.
Рецензент: доктор фіз.-мат. наук, доцент Скрипник Н.В.

Рекомендовано до захисту:
Протокол засідання кафедри
№ ___ від «_____» _____ р.
Завідувач кафедри
_____ _____

Захищено на засіданні ЕК № _____
Протокол № ____ від «____» ____ р.
Оцінка _____ / _____ / _____
Голова ЕК
_____ _____

Одеса — 2022 р.

ЗМІСТ

Вступ 3

1 Алгоритми натхненні природою 5
1.1 Алгоритм диференціальної еволюції 5
1.2 Алгоритм імітації відпалу 7
1.3 Алгоритм гармонічного пошуку 8

2 Оптимізація гіперпараметрів 11
2.1 Гіперпараметри і параметри 11
2.2 Найпоширеніші методи підбору гіперпараметрів 11
2.2.1 Налаштування вручну 12
2.2.2 Пошук у сітці 12
2.2.3 Випадковий пошук 12
2.2.4 Байєсівська оптимізація 13
2.2.5 Оптимізація на основі градієнта 13
2.2.6 Генетичний алгоритм 13
2.2.7 Популяційне навчання 13
2.3 Основні інструменти 14
2.4 Природні алгоритми в задачі оптимізації гіперпараметрів . . 15

3 Застосування 16
3.1 Набори даних 16
3.1.1 Титанік 16
3.1.2 Телекомунікаційний відтік 17
3.2 Обчислювальний експеримент 18
3.3 Аналіз результатів 18
3.3.1 Титанік 19
3.3.2 Телекомунікаційний відтік 19

Висновки 21

Перелік використаних джерел 22

Додатки 23

ВСТУП

Машинне навчання — це галузь штучного інтелекту, яка спрямована на використання даних і алгоритмів для імітації навчання людини, поступово підвищуючи точність. Сфера машинного навчання виросла із традиційних спільнот статистики та штучного інтелекту. ML може відігравати ключову роль у широкому спектрі важливих додатків, таких як інтелектуальний аналіз даних, обробка природної мови, розпізнавання зображень, експертні системи тощо.

Оптимізація моделі є однією з найскладніших проблем у реалізації рішень машинного навчання. Цілі галузі машинного навчання та теорії глибокого навчання були спрямовані на оптимізацію моделей. Оптимізація гіперпараметрів у машинному навчанні має на меті знайти гіперпараметри даного алгоритму машинного навчання, які забезпечують найкращу продуктивність, виміряну на наборі перевірки[1]. Гіперпараметри - це всі параметри моделі, які не оновлюються під час навчання і використовуються для налаштування будь-якої моделі. Налаштування гіперпараметрів має вирішальне значення, оскільки вони контролюють загальну поведінку моделі машинного навчання.

Існує багато способів оптимізації гіперпараметрів. Серед них пошук по сітці, випадковий пошук, ручний підбір, оптимізація Баєса тощо. Все більшої популярності набувають алгоритми, натхненні природою, для вирішення проблеми пошуку гіперпараметрів. Алгоритми, натхненні природою, дійсно потужні і намагаються знайти найкраще рішення проблеми, однак не гарантують, що найкраще рішення буде знайдено взагалі.

Алгоритми, натхненні природою, — це набір нових методологій і підходів вирішення проблем, отриманих від природних процесів. Деякі з популярних прикладів натхненних природою алгоритмів оптимізації включають: генетичний алгоритм, оптимізацію рою частинок , алгоритм пошуку зозулі, оптимізацію колонії мурах тощо.

Мета дослідження: вивчення та дослідження алгоритмів, натхненних природою, для підбору гіперпараметрів задачі класифікації.

Предмет дослідження: застосування природних алгоритмів для опти-

мізації пошуку рішення прикладних задач.

Об'єкти дослідження: алгоритми, натхненні природою - алгоритм диференційної еволюції, імітації обпалу та гармонічного пошуку.

Методи дослідження: проведення експериментів за допомою фреймворку Keras, аналіз результатів.

В рамках дипломної роботи була розглянута задача класифікації за допомогою нейронної мережі. Оптимізацію моделі було здійснено за допомогою алгоритмів, натхненних природою - алгоритмів диференційної еволюції, імітації обпалу та гармонічного пошуку. Завдяки часовим замірам роботи алгоритмів, результати було порівняно та проаналізовано.

РОЗДІЛ 1

АЛГОРИТМИ НАТХНЕННІ ПРИРОДОЮ

Алгоритми, натхненні природою, — це набір нових методологій і підходів вирішення проблем, отриманих від природних процесів. Деякі з популярних прикладів алгоритмів оптимізації, натхнених природою, включають: генетичний алгоритм, оптимізацію рою частинок, алгоритм пошуку зозули, оптимізацію колонії мурах тощо.

За останні кілька десятиліть різні дослідники запропонували велику кількість натхненних природою алгоритмів. Деякі з цих алгоритмів виявилися дуже ефективними в порівнянні з іншими класичними методами оптимізації, адже дають можливість отримати оптимальні рішення для більш широкого кола проблем за досить практичний час.

1.1 Алгоритм диференціальної еволюції

Алгоритм диференціальної еволюції (Differential Evolution Algorithm) належить до широкого сімейства еволюційних обчислювальних алгоритмів. Вперше він був представлений у 1996 та 1997 роках Р.Сторном та К.Прайсом відповідно.

Подібно до інших популярних підходів прямого пошуку, таких як генетичні алгоритмиї, алгоритм диференціальної еволюції починається з початкової сукупності кандидатських рішень. Ці рішення-кандидати багаторазово вдосконалюються шляхом введення мутацій у популяцію, кросоверу та збереження найбільш підходящих рішень-кандидатів за значеннями цільової функції[2].

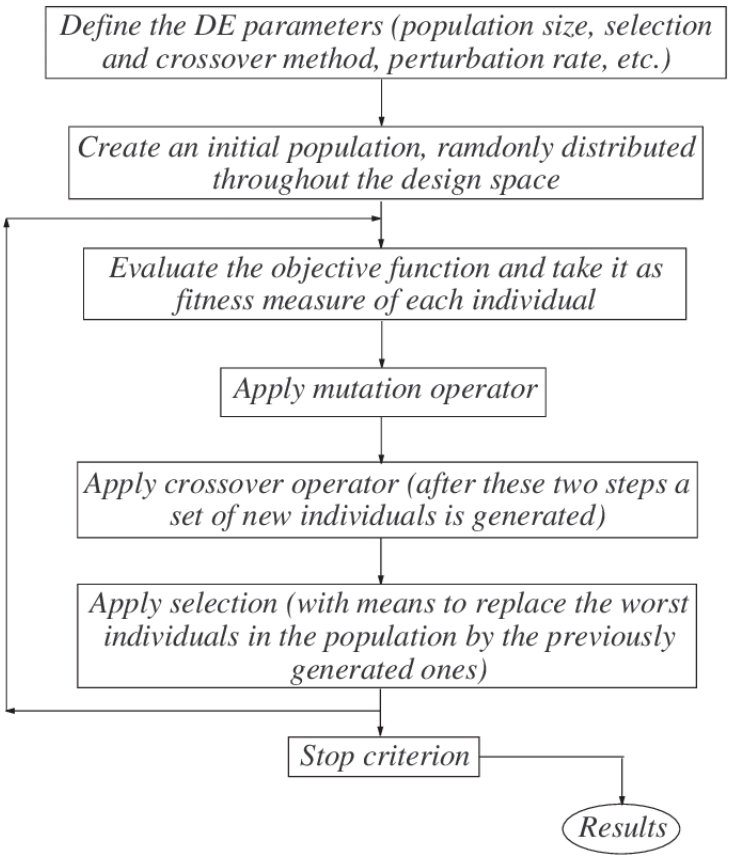

Рисунок 1.1. Схема алгоритму диференціальної евелюції

Мутацію в DEA можна розглядати як узагальнений шаблон пошуку в будь-якому випадковому напрямку $(x_p x_q)$ за

$$x_i = x_r + F(x_p x_q)$$

де F – диференціальна вага в діапазоні $[0, 2]$. Тут r, p, q – відмінні одне від одного цілі числа, згенеровані випадковою перестановкою.

Оператор кросоверу залежить від ймовірності $C_r \in [0,1]$. Фактично сам процес кросоверу часто буває біноміальний або експоненційний.

Оператор відбору не відрізняється від випадку генетичного алгоритму. Він полягає у виборі найбільш придатних за функцією якості особин.

Основною перевагою DEA є той факт, що він має лише три параметри керування, які користувачеві алгоритму необхідно налаштувати. Сюди входять розмір популяції NP, де $NP \geq 4$, коефіцієнт мутації (або коефіцієнт масштабування) $F \in [0, 2]$ і ймовірність перехресного переходу (або контрольний параметр кросовера) $CR \in [0, 1]$.

1.2 Алгоритм імітації відпалу

Алгоритм імітації відпалу (Simulated Annealing Algorithm) — це стохастичний алгоритм глобальної пошукової оптимізації. Він був вперше описаний Д.Кікпатріком у 1983 році.

Алгоритм розроблений на основі відпалу в металургії — техніки, при якій метал нагрівають до високої температури та повільно охолоджують для покращення його фізичних властивостей. Коли метал гарячий, молекули випадковим чином швидко перебудовуються. Коли метал починає остигати, процес перегрупування відбувається набагато повільніше. Зрештою, отриманий метал буде бажаним [3].

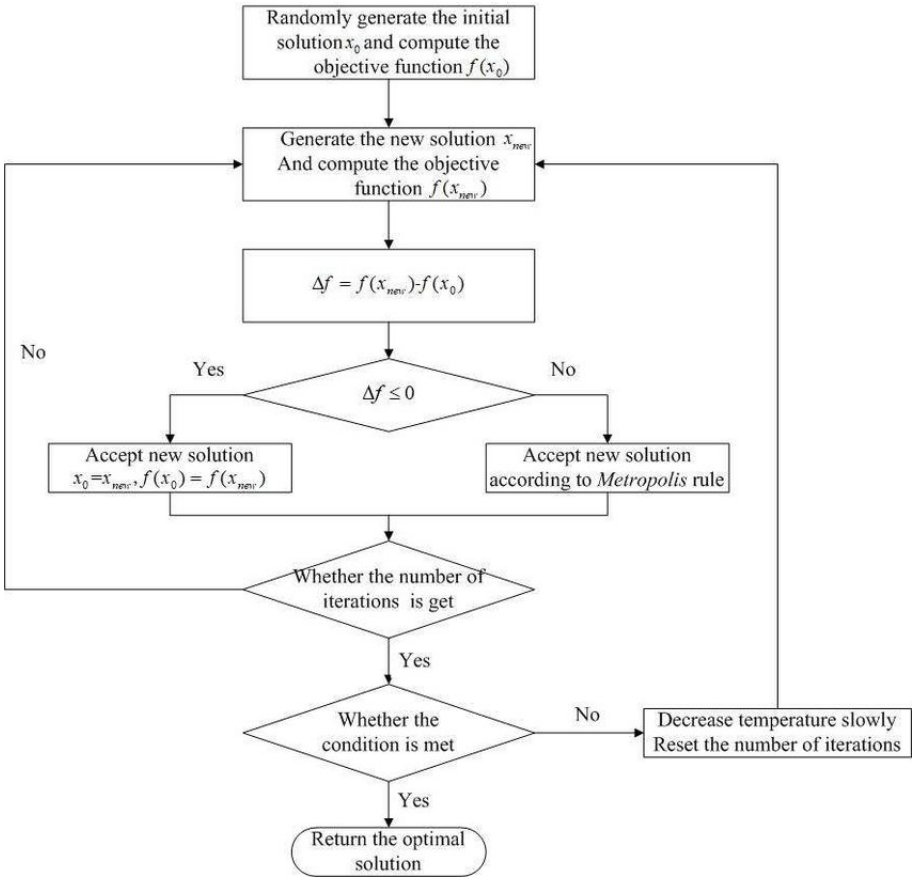

Рисунок 1.2. Схема алгоритму імітації відпалу

В даному алгоритмі прийняття поточного рішення приймається за допомогою критерію Метрополіса:

$$p(x_i \to x_j | x(t) = x_i) = \exp[-\frac{1}{T(t)} \max\{0, f(x_j) - f(x_i)\}]$$

Тут f - цільова функція, T - аналог температури в системі відпалу.

Основним оператором є створення нових рішень шляхом випадкових блукань, і, отже, рандомізація діє як механізм мутації або дослідницького пошуку. Відбір досягається шляхом перевірки, чи стає рішення краще. Оскільки, імітований відпал не є еволюційним алгоритмом, в цьому алгоритмі немає оператора кросовера [4].

Процес зупиняється після досягнення певної температури. Речовина охолола у точці з мінімальною енергією. Щоб не застрягти в локальному мінімумі й знайти мінімум глобальний, необхідно час від часу підвищувати енергію (цільву функцію) системи. У цьому загальна тенденція до пошуку найменшої енергії зберігається.

1.3 Алгоритм гармонічного пошуку

Алгоритм гармонічного пошуку (Harmony Search Algorithm) — це музичний алгоритм, розроблений Зонг Ву Гемом у 2001 році. Цей алгоритм є метаевристичним. У стандартному HSA рішення представлені в термінах сукупності гармоній музиканта, що грає музичний твір, використовуючи наступні три правила:

1) зіграти будь-який відомий музичний твір по пам'яті
2) зіграти щось подібне до відомого твору
3) створювати нові або випадкові ноти.

HS використовує головним чином мутацію та відбір, тоді як кросовер явно не використовується. Перше правило відповідає відбору або елітарності, а друге і третє правила є мутацією [4].

На концерті оркестру після того, як гобой грає ноту A , інші інструменти випадковим чином грають будь-яку висоту звуку за межами діапазону відтворення. Аналогічно, алгоритм HSA спочатку імпровізує багато випадкових гармоній. Кількість випадкових гармоній має бути не менше кількісті векторів рішення, які одночасно обробляються в алгоритмі.

Гармонійну пам'ять музиканта можна розглядати як матрицю:

$$\left[\begin{array}{cccc|c} x_1^1 & x_2^1 & \dots & x_n^1 & f(x^1) \\ x_1^2 & x_2^2 & \dots & x_n^2 & f(x^2) \\ \vdots & \dots & \dots & \dots & f(x^2) \\ x_1^3 & x_2^3 & \dots & x_n^3 & f(x^3) \end{array}\right]$$

Музикант намагається змінити різні тони, поки не знайдеться ідеальна гармонія. У задачі оптимізації оператори зазнають різних варіацій; якщо результати варіації сприятливі, то пам'ять оновлюється шляхом додавання цього до пам'яті та видалення небажаного. Під час створення музики відбувається одна з цих трьох подій:

1) Вибирається один із збережених тонів із набору різних з гармонійної пам'яті.
2) Вибирається будь-який суміжний тон, який ближче до одного із збережених (регулювання висоти)
3) Вибирається будь-який довільний тон, який підпадає під стандартний діапазон (випадковий вибір) [5]

Основними компонентами HSA є розмір гармонійної пам'яті, швидкість розгляду пам'яті гармонії, швидкість регулювання висоти звуку і критерії зупинки.

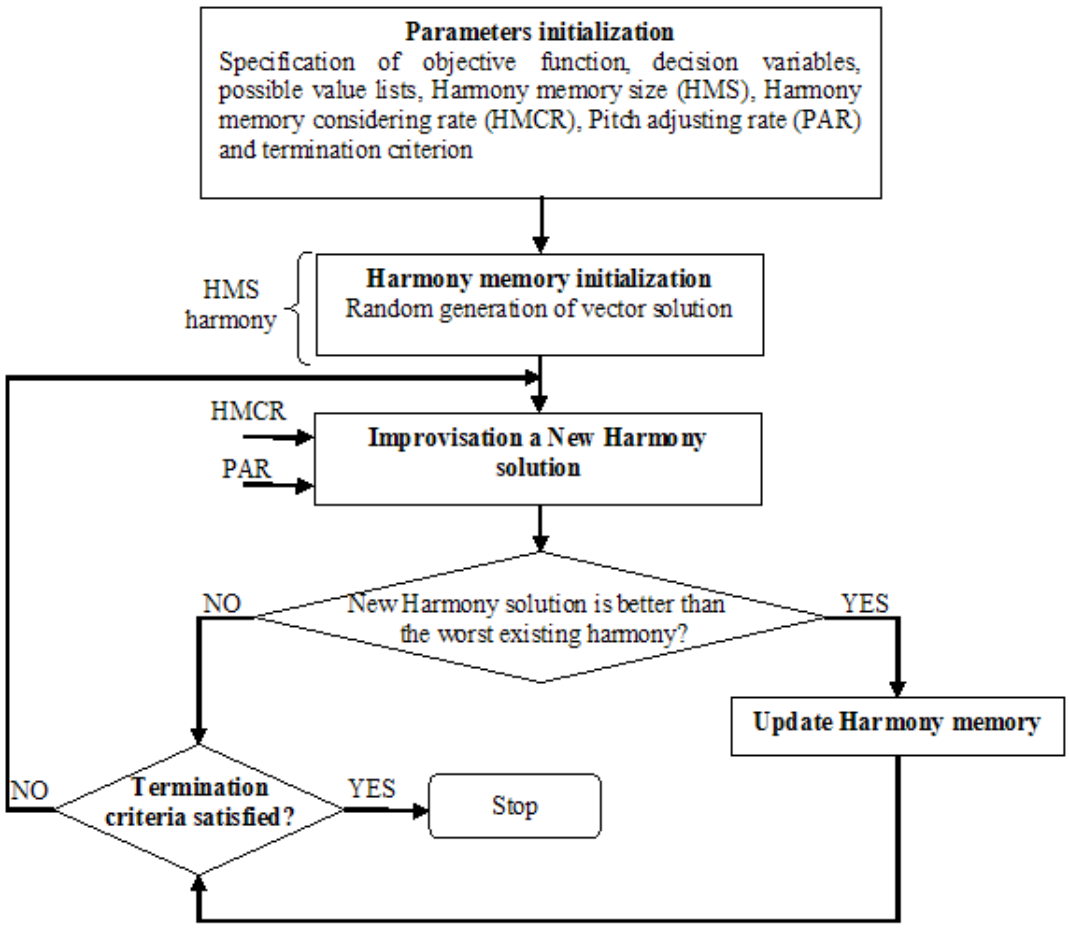

Рисунок 1.3. Схема алгоритму гармонічного пошуку

Процес еволюції алгоритму гармонічного пошуку припиняється, якщо досягається умова припинення. Умову припинення можна визначити за загальною кількістю досягнутих на даний момент імпровізацій.

РОЗДІЛ 2

ОПТИМІЗАЦІЯ ГІПЕРПАРАМЕТРІВ

У ході створення моделі машинного навчання розробник постійно має примати рішення — які дані аналізувати, який підхід використовувати, який тип результату генерувати тощо. Ключовими частинами моделі машинного навчання, що визначають її структуру та поведінку, є її гіперпараметри.

Гіперпараметр — це аргумент конфігурації моделі, зазначений розробником для керування процесом навчання для певного набору даних.

Налаштування (оптимізація) гіперпараметрів є процесом визначення правильної комбінації гіперпараметрів, яка максимізує продуктивність моделі. Результатом оптимізації гіперпараметрів є єдиний набір високоефективних гіперпараметрів, які можна використовувати для налаштування моделі.

2.1 Гіперпараметри і параметри

Моделі машинного навчання також мають параметри - внутрішні коефіцієнти, що встановлені шляхом навчання або оптимізації моделі на наборі навчальних даних.

Головна різниця між параметрами та гіперпараметрами полягає в тому, що параметри вивчаються автоматично, а гіперпараметри встановлюються вручну, щоб допомогти керувати процесом навчання.

2.2 Найпоширеніші методи підбору гіперпараметрів

Існує кілька методів оптимізації гіперпараметрів, вони мають різні переваги та недоліки при застосуванні до різних типів задач. У наступних підрозділах були розглянуті одні з найпоширеніших методів налаштування

гіперпараметрів моделі.

2.2.1 Налаштування вручну

Ручне налаштування гіперпараметрів передбачає експериментування з різними наборами гіперпараметрів вручну. Використовуючи ручний пошук, гіперпараметри моделі обираються на основі наших суджень або досвіду користувача. Потім модель навчається й проводиться оцінка її якості. Цей цикл повторюється до тих пір, поки не буде досягнута задовільна якість .

2.2.2 Пошук у сітці

Пошук у сітці (Grid Search) є традиційним методом налаштування гіперпараметрів. Він виконує вичерпний пошук за набором гіперпараметрів, зазначеним користувачами. Цей підхід є найбільш простим і веде до найбільш точних прогнозів. Використовуючи цей метод налаштування, користувачі можуть знайти оптимальну комбінацію. Пошук у сітці повільніше в порівнянні з іншими методами, але в цілому він може бути ефективнішим, оскільки може проходити через весь простір пошуку [6].

2.2.3 Випадковий пошук

Випадковий пошук (Random Search) можна назвати основним удосконаленням пошуку в сітці. Створюється сітка можливих значень гіперпараметрів. Кожна ітерація застосовує випадкову комбінацію гіперпараметрів з цієї сітки, записує продуктивність і, нарешті, повертає комбінацію гіперпараметрів, яка забезпечує найкращу продуктивність.

Генерування випадкових вибірок є тривіальним з точки зору обчислень і не займає багато пам'яті, тому може бути ефективним створити велику вибірку вхідних даних, а потім оцінити їх. Кожен зразок є незалежним, тому зразки можна оцінювати паралельно, якщо це необхідно для прискорення процесу.

Основним недоліком випадкового пошуку є те, що кращий набір

гіперпараметрів може бути пропущеним в результаті створення випадкових виборок.

2.2.4 Байєсівська оптимізація

Байєсівська оптимізація стала ефективним інструментом для налаштування гіперпараметрів алгоритмів машинного навчання, точніше, для складних моделей, таких як глибокі нейронні мережі.

Цей метод послідовний. Він використовує результати попередніх наборів гіперпараметрів для покращення наступного процесу пошуку. Байєсівський пошук скорочує час оптимізації, особливо для моделей, які навчаються на великій кількості даних [8].

2.2.5 Оптимізація на основі градієнта

Оптимізація на основі градієнта — це методологія оптимізації кількох гіперпараметрів , заснована на обчисленні градієнта критерію вибору моделі машинного навчання щодо гіперпараметрів. Цю методологію налаштування гіперпараметрів можна застосувати, коли задовольняються деякі умови диференційності та безперервності критерію навчання [6].

2.2.6 Генетичний алгоритм

Genetic Algorithms намагається застосувати механізми природного відбору до контекстів машинного навчання. Вони натхненні дарвінівським процесом природного відбору, тому їх також зазвичай називають еволюційними алгоритмами.

2.2.7 Популяційне навчання

Ця техніка є гібридом двох найбільш часто використовуваних методів пошуку: випадкового пошуку та ручного налаштування, що застосовуються до моделей нейронної мережі. Починається з навчання багатьох нейронних

мереж паралельно з випадковими гіперпараметрами. Але ці мережі не є повністю незалежними одна від одної [9].

2.3 Основні інструменти

Бібліотека машинного навчання Python scikit-learn з відкритим кодом надає методи для налаштування гіперпараметрів моделі. Зокрема, він надає RandomizedSearchCV для випадкового пошуку та GridSearchCV для пошуку в сітці. Обидва методи оцінюють моделі для заданого вектора гіперпараметрів за допомогою перехресної перевірки, отже, суфікс «CV» кожного імені класу.

Optuna — це програма оптимізації гіперпараметрів, яка здатна легко реалізувати різні найсучасніші методи оптимізації для швидкого виконання оптимізації гіперпараметрів з високою продуктивністю. За налаштуванням Optuna реалізує байєсівський алгоритм оптимізації (TPE), але його можна легко переключити на інші існуючі алгоритми в пакеті[7].

Hyperopt є одним із найпопулярніших доступних пакетів налаштування гіперпараметрів. Hyperopt дозволяє користувачеві описати простір пошуку, в якому користувач очікує найкращих результатів, що дозволяє алгоритмам в hyperopt здійснювати пошук більш ефективно.

Spearmint — це програмний пакет, який також виконує байєсівську оптимізацію. Програмне забезпечення призначене для автоматичного проведення експериментів таким чином, що ітераційно налаштовує ряд параметрів, щоб мінімізувати деякі цілі за якомога менше ітерацій.

Щоб реалізувати генетичні алгоритми в Python, можна використовувати бібліотеку TPOT Auto Machine Learning . TPOT побудований на основі бібліотеки scikit-learn, і його можна використовувати для завдань регресії або класифікації.

2.4 Природні алгоритми в задачі оптимізації гіперпараметрів

Алгоритми, натхненні природою, в задачах оптимізації є високоефективними у пошуку оптимізованих рішень багатовимірних і мультимодальних задач. Ці алгоритми використовують стохастичний підхід, щоб знайти найкраще рішення у великому просторі пошуку задачі. Налаштування гіперпараметрів можна вирішити з меншими обчислювальними зусиллями та часовою складністю.

РОЗДІЛ 3

ЗАСТОСУВАННЯ

3.1 Набори даних

В даній дипломній роботі експерименти проводилися на відомих класичних датасетах - "Titanic"та "Telecom Churn Dataset".

3.1.1 Титанік

У 1912 році корабель «Титанік» під час свого першого рейсу зіткнувся з айсбергом і затонув. Це призвело до смерті більшості його пасажирів і екіпажу. У даному датасеті подано інформацію про долю пасажирів «Титаніка» відповідно до економічного стану (класу), статі, віку тощо.

Навчальний набір містить 891 приклад і 11 ознак + цільова змінна (факт виживання). Цільова змінна - стовпець "Survived"містить значення 0 чи 1, що відповідає факту виживання пасажира у катастрофі. Серед ознак наявні:

1) PassengerId - унікальний ідентифікатор кожного пасажира (числовий)
2) Pclass - клас білета
3) Name - ім'я
4) Sex - стать
5) Age - вік пасажира в роках
6) SibSp - кількість братів, сестер, чоловіків та дружин пасажира на борту Тітаніка
7) Parch - кількість батьків та дітей пасажира на борту Тітаніка
8) Fare - тариф
9) Cabin - номер каюти
10) Embarked - порт посадки

Метою аналізу є встановлення зв'язку між характеристиками пасажира та його порятунком.

3.1.2 Телекомунікаційний відтік

Прогнозування відтоку клієнтів має вирішальне значення для телекомунікаційних компаній , щоб мати можливість ефективно утримувати клієнтів. Залучати нових клієнтів дорожче, ніж утримувати наявних. З цієї причини великі телекомунікаційні корпорації прагнуть розробити моделі, щоб передбачити, які клієнти, швидше за все, можуть піти, і вжити відповідних заходів.

Набір даних "Telecom Churn Dataset"містить дев'ятнадцять стовпців, які вказують на характеристики клієнтів вигаданої телекомунікаційної корпорації. Стовпець Churn вказує, чи відбув клієнт протягом останнього місяця, чи ні. До класу "No"входять клієнти, які не залишили компанію минулого місяця, а до класу "Yes"– клієнти, які вирішили припинити відносини з компанією. Набір даних містить 19 незалежних змінних :

1) Gender - стать
2) SeniorCitizen - чи є клієнт пенсіонером
3) Partner - наявність партнера
4) Dependents - наявність залежних осіб
5) Tenure - кількість місяців, протягом яких клієнт перебував у компанії
6) Contract - поточний тип контракту клієнта
7) PaperlessBilling - наявність безпаперового рахунку
8) PaymentMethod - спосіб оплати клієнта
9) MontlyCharges - сума, що стягується з клієнта щомісяця
10) TotalCharges - загальна сума, стягнена з клієнта
11) PhoneService - наявність у клієнта телефонної послуги
12) MultipleLines - наявність у клієнта кількох номерів
13) InternetServices - наявність підключення клієнту на послугу Інтернету в компанії
14) OnlineSecurity - наявність у клієнта онлайн-безпеки
15) OnlineBackup - наявність у клієнта резервного копіювання в режимі онлайн
16) DeviceProtection - наявністьу клієнта захисту пристрою
17) TechSupport - наявність у клієнта технічної підтримки
18) StreamingTV - наявність у клієнта потокового телебачення

19) StreamingMovies - наявність у клієнта потокових фільмів

Метою аналізу є встановлення зв'язку між характеристиками клієнта та відтоком.

3.2 Обчислювальний експеримент

Основна мета цієї роботи – оптимізувати пошук гіперпараметрів моделі за допомогою алгоритмів, натхненних природою. Експерименти проводилися на задачах бінарної класифікації. В якості класифікатора виступає нейронна мережа з 2 шарами та сігмоїдою як функцією активації на останньому кроці.

Цільовою функцією виступає міра точності моделі. Це першого датасету була обрана метрика accuracy, а для другого - F1. Це пов'язано з тим, що в Telecom Churn Dataset класи цільової змінної не збалансовані.

Accuracy — це показник, який загалом описує ефективність моделі в усіх класах. Це корисно, коли всі класи мають однакову важливість. Він розраховується як відношення між кількістю правильних прогнозів до загальної кількості передбачень.

Оцінка F1 – це середнє зважене значення метрик Precision та Recall. Таким чином, ця оцінка враховує як помилкові позитивні, так і помилкові негативи.

3.3 Аналіз результатів

Пошук гіперпараметрів відбувався за 10 епох та з кількістю особин у 50 штук. Шукані гіперпараметри:

1) Швидкість навчання
2) Кількість вузлів прихованого шару
3) Кількість епох
4) Частка вхідних одиниць, які потрібно викинути аби запобігти перенавчання
5) Оптимайзер

6) Функція активації

Код розв'язування задачі наведений в Додатках.

3.3.1 Титанік

	Harmony Search	Simulated Annealing	Differential Evolution
Time, s	1454.6	227.04	1420.93
Accuracy	0.853	0.8466	0.873
Best Parameters	[0.05, 50, 100, 0.3, 'Adam', 'relu']	[0.01, 20, 50, 0.3, 'Adam', 'relu']	[0.001, 50, 50, 0.1, 'RMSprop', 'relu']

Застосовуючи дефолтні значення гіперпараметрів точність становить - 0,56.

Всі використані алгоритми непогано впоралися з задачею підбору гіперпараметрів. Точність моделі значно покращилася в порівнянні з точністю зі стандартими гіперпараметрами.

- Алгоритм гармонічного пошуку за найдовший час забеспечив середню з отриманих точність.
- Алгоритм імітації відпалу дуже швидко впорався з поставленою задачею, але отримана якість моделі виявилася найгіршою в порівнянні з іншими моделями. Можливо, збільшивши кількість ітерацій, можна було б забезпечити більш високу точність.
- Алгоритм диференціальної еволюції відпрацьовував трохи швидше ніж гармонічний пошук, але отримана якість моделі набагато вища в порівнянні з іншими алгоритмами.

3.3.2 Телекомунікаційний відтік

	Harmony Search	Simulated Annealing	Differential Evolution
Time, s	5897.4	936.12	4847
F1-score	0.913	0.92	0.913
Best Parameters	[0.001, 100, 50, 0.3, 'Nadam', 'relu']	[0.01, 50, 100, 0.3, 'RMSprop', 'relu']	[0.1, 100, 100, 0.5, 'Adagrad', 'tanh']

Застосовуючи дефолтні значення гіперпараметрів F1-score становить - 0,66.

Використані алгоритми добре впоралися з задачею підбору гіперпараметрів. Якість моделі значно покращилася в порівнянні зі стандартими параметрами.

- Алгоритм гармонічного пошуку відпрацьовував за найдовший час. Отримана точність моделі співпала з результатом диференціальної еволючії.
- Алгоритм імітації відпалу порівняно швидко впорався з поставленою задачею та зміг підбрати найкращий набір гіперпараметрів серед інших алгоритмів.
- Алгоритм диференціальної еволюції за середній час забеспечив меншу з отриманих F1-score, але різниця з алгоритмом, що впорався найкраще, невелика.

Найбільше часу потребє процес навчання моделі. В алгоритмі імітації відпалу немає потреби кожної ітерації перераховувати або сортувати значення функції якості (що неможливо без тренування). Саме завдяки цьому алгоритм іміцації обпалу виявляється таким швидким.

ВИСНОВКИ

Під час виконання дипломної роботи було вивчено алгоритми, натхненні природою - алгоритм імітації відпалу, гармонічного пошуку та диференційної еволюції. Усі вищезгадані алгоритми були застосовані в задачі пошуку оптмальних гіперпаматерів моделі машинного навчання. Аналіз результатів був здійснений на основі датасетів "Титанік"та "Телекомунікаційний відтік". Найкращим серед усіх алгоритмів на першому датасеті виявися алгоритм диференціальної еволюції, на другому - алгоритм імітації відпалу.

Дану роботу можна було б вдосконалити за рахунок поширення написаних алгортмів для інших задач машинного навчання - наприклад, для обробки природньої мови чи для розпізнавання зображень. Більш того, можна було б використати інші алгоритми, натхнені природою й порівняти результати з вже отриманими.

Задача пошуку гіперпараметрів є однією з найскладніших та найважливіших задач в машинному навчанні. Цей процес досить трудомісткий. Саме тому важливо приділяти оптимізації гіперпараметрів належної уваги та досліджувати нові методи вирішення цієї проблеми.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Chauhan, N. S. (2020). Hyperparameter Optimization for Machine Learning Models. https://www.kdnuggets.com/2020/05/hyperparameter-optimization-machine-learning-models.html
2. Stefania C. (2021). Differential Evolution from Scratch in Python. https://machinelearningmastery.com/differential-evolution-from-scratch- in-python/
3. Brownlee J. (2021). Simulated Annealing From Scratch in Python. https://machinelearningmastery.com/simulated-annealing-from-scratch- in-python/
4. Yang X.-S. (2014). Nature-Inspired Optimization Algorithms. https://msulaiman.org/onewebmedia/Xin-She-Yang-Auth.-Nature-Inspired-Optimization-Algorithms.pdf
5. Dubey M. (2021). A Systematic Review on Search Algorithm: Theory, Literature, and Harmony Applications. https://www.hindawi.com/journals/mpe/2021/5594267/
6. Choudhure A. (2021). Top 8 Approaches For Tuning Hyperparameters Of Machine Learning Models. https://analyticsindiamag.com/top-8-approaches- for-tuning-hyperparameters-of-machine-learning-models/
7. Lim Ye. (2021). State-of-the-Art Machine Learning Hyperparameter Optimization with Optuna. https://towardsdatascience.com/state-of- the-art-machine-learning-hyperparameter-optimization-with-optuna- a315d8564de1
8. Dilmegani C. (2021). Hyperparameter Optimization: Methods And Top Tools in 2022. https://research.aimultiple.com/hyperparameter- optimization/
9. Jaderberg M. (2017). Population Based Training of Neural Networks. https://arxiv.org/abs/1711.09846
10. Brownlee J. (2020). Hyperparameter Optimization With Random Search and Grid Search. https://machinelearningmastery.com/hyperparameter-optimization-with-random-search-and-grid-search/

ДОДАТКИ

```
#Parameters values
optimizer = [keras.optimizers.SGD,
             keras.optimizers.Adam,
             keras.optimizers.Nadam,
             keras.optimizers.RMSprop,
             keras.optimizers.Adagrad]
learning_rate = [0.001, 0.01, 0.05, 0.1]
activation = ['relu', 'linear', 'tanh']
epochs = [50, 100]
dr = [0.1, 0.3, 0.5]
nodes = [5, 10, 20, 50, 100]
params = [learning_rate, nodes, epochs, dr, optimizer, activation]

genes_num = len(params)
size = 50
epoches = 10

#General
def createModel(optimizer = 'sgd',
                activation = 'relu',
                dr = 0.1,
                learning_rate = 0.1,
                nodes = 5):
    model = Sequential()
    model.add(Dense(nodes, activation=activation))
    model.add(Dropout(dr))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(loss='binary_crossentropy',
                  optimizer=optimizer(learning_rate=learning_rate),
                  metrics=['accuracy'])
    return model

def createEntity():
    entity = []
    for i in range(genes_num):
        num = np.random.randint(len(params[i]))
        entity.append(params[i][num])
    return entity

def createInitial():
    population = []
    for i in range(size):
        population.append(createEntity())
    return population

def fitness(entities):
    fit_values = []
    for i in range(size):
        fit_values.append(fit(entities[i]))
    return fit_values

def fit(value):
    model = KerasClassifier(build_fn=createModel,
                            optimizer = value[4],
                            activation = value[5],
                            dr = value[3],
                            learning_rate = value[0],
                            nodes = value[1])
```

```
    model.fit(X_train , y_train, epochs = value[2], verbose=0)
    y_pred = np.around(model.predict(X_test))
    return metrics.accuracy_score(y_test, y_pred)

def generateAnotherFromArr(size, arr):
    num = np.random.randint(size)
    while num in arr:
        num = np.random.randint(size)
    return num

# Simulated annealing
def probability(fit_new, fit_old, t):
    return np.exp((fit_new - fit_old) / t)

def simulatedAnnealing():
    start = time.time()
    print()
    print(time.time() - start, "~~~~~~~~~~~~~~~~Simulated Annealing~~~~~~~~~~~~~~~~")
    t = 1
    x_index = 0
    entities = createInitial()
    print("Initial: ", entities)
    for i in range(epoches):
        print(time.time() - start, "---------Epoch " + str(i+1) + " ---------")
        t = t * 0.99
        y_index = generateAnotherFromArr(size, [x_index])
        y = entities[y_index]
        fit_x = fit(entities[x_index])
        fit_y = fit(y)
        if fit_y >= fit_x:
            entities[x_index] = y
            x_index = y_index
        else:
            prob = probability(fit(entities[x_index]), fit(y), t)
            if np.random.random() < prob:
                entities[x_index] = y
        print(entities)

    print()
    print("Final entities: ", entities)
    fit_entities = fitness(entities)
    print("Fit values: ", fit_entities)
    sorted_entities_indexes = np.argsort(fit_entities[:])
    print("Best accuracy: ",
          fit_entities[sorted_entities_indexes[-1]],
          " with params ",
          entities[sorted_entities_indexes[-1]])
    print(time.time() - start, "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~")

#Differential evolution
def mutation(population, mutation_prob, diff_weight):
    mutate_values = []
    for i in range(size):
        if np.random.random() < mutation_prob:
            f_i = generateAnotherFromArr(size, [i])
            s_i = generateAnotherFromArr(size, [i, f_i])
            th_i = generateAnotherFromArr(size, [i, f_i, s_i])
            new = population[i]
            for j in range(genes_num):
                index = int(np.around(f_i + diff_weight*abs(s_i-th_i))) % len(params[j])
                new[j] = params[j][index]
```

```
            mutate_values.append(new)
        else:
            mutate_values.append(population[i])
    return mutate_values

def selection(population, selection_thres):
    fit_population = fitness(population)
    new_population = []
    best_index = 0
    new_population.append(population[best_index])
    for i in range(1, size):
        if fit_population[i] <= fit_population[best_index]:
            new_population.append(population[best_index])
        elif fit_population[i] > fit_population[best_index]:
            best_index = i
            new_population.append(population[i])
    return new_population

def crossover(selected_population, crossover_prob):
    new_population = []
    for i in range(size):
        f_index = np.random.randint(len(selected_population))
        s_index = generateAnotherFromArr(len(selected_population), [f_index])
        entity = []
        for j in range(genes_num):
            if np.random.random() < crossover_prob:
                entity.append(selected_population[f_index][j])
            else:
                entity.append(selected_population[s_index][j])
        new_population.append(entity)
    return new_population

def differentialEvolution():
    start = time.time()
    print()
    print(time.time() - start, "~~~~~~~~~~~~~~~~Differential Evolution~~~~~~~~~~~~~~~~")
    mutation_prob = 0.1
    selection_thres = 0.6
    crossover_prob = 0.9
    diff_weight = 0.8
    population = createInitial()
    print("Initial: ", population)
    for i in range(epoches):
        print(time.time() - start, "---------Epoch " + str(i+1) + " ---------")
        population = mutation(population,  mutation_prob, diff_weight)
        population = crossover(population, crossover_prob)
        population = selection(population, selection_thres)
        print(population)
    print()
    print("Final population: ", population)
    fit_entities = fitness(population)
    print("Fit values: ", fit_entities)
    sorted_entities_indexes = np.argsort(fit_entities[:])
    print("Best accuracy: ",
          fit_entities[sorted_entities_indexes[-1]],
          " with params ",
          population[sorted_entities_indexes[-1]])
    print(time.time() - start, "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~")


#Harmony search
```

```
def harmonySearch():
    start = time.time()
    print()
    print(time.time() - start, "~~~~~~~~~~~~~~~~Harmony Search~~~~~~~~~~~~~~~~")
    hmcr = 0.9
    par = 0.2
    entities = createInitial()
    print("Initial: ", entities)
    for i in range(epoches):
        print(time.time() - start, "---------Epoch " + str(i+1) + " ---------")
        fit_entities = fitness(entities)
        sorted_entities_indexes = np.argsort(fit_entities[:])
        worst_fit = fit_entities[sorted_entities_indexes[0]]
        for j in range(size):
            if np.random.random() < hmcr:
                if np.random.random() < par:
                    rand = np.random.random()
                    for gene in range(genes_num):
                        b = len(params[gene])
                        entities[j][gene] = params[gene][int(np.around(gene + b * rand)) % len(params[gene])]
            else:
                entities[j] = createEntity()
            if worst_fit < fit_entities[j]:
                entities[sorted_entities_indexes[0]] = entities[j]
                fit_entities[sorted_entities_indexes[0]] = fit_entities[j]
                sorted_entities_indexes = np.argsort(fit_entities[:])
                worst_fit = fit_entities[sorted_entities_indexes[0]]
        print(entities)
    print()
    print("Final entities: ", entities)
    fit_entities = fitness(entities)
    print("Fit values: ", fit_entities)
    sorted_entities_indexes = np.argsort(fit_entities[:])
    print("Best accuracy: ",
          fit_entities[sorted_entities_indexes[-1]],
          " with params ",
          entities[sorted_entities_indexes[-1]])
    print(time.time() - start, "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~")

harmonySearch()
simulatedAnnealing()
differentialEvolution()
```