МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«До захисту допущено»
В.о. завідувача кафедри
Ігор ШЕЛЕХОВ
_____
(підпис)
2024 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавр

зі спеціальності 122 - Комп'ютерних наук,
освітньо-наукової програми «Інформатика»
на тему: «Інформаційна система дослідження властивостей білків з використанням машинного навчання»
здобувача групи ІН-01 Проценка Ярослава Сергійовича

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

_____ Ярослав ПРОЦЕНКО
(підпис)

Керівник,
кандидат фізико-математичних наук,
доцент Надія ТИРКУСОВА _____
(підпис)

Суми – 2024

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня магістр

зі спеціальності 122 - Комп'ютерних наук, освітньо-наукової програми «Інформатика»
здобувача групи ІН-11 Проценка Ярослава Сергійовича

1. Тема роботи: «Інформаційна система дослідження властивостей білків з використанням машинного навчання»
затверджую наказом по СумДУ від «22» квітня 2024 р. № 0414-IV
2. Термін здачі здобувачем кваліфікаційної роботи до 01 червня 2024 року
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
1) Аналіз проблеми предметної області, постановка й формування завдань дослідження. 2) Огляд алгоритмів, що використовуються для прогнозування властивостей білків. 3) Розробка інформаційної системи дослідження властивостей білків. 4) Аналіз результатів.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____
6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «06» травня 2024 р.

Завдання прийняв до виконання _____ (підпис) Керівник _____ (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	Аналіз проблеми предметної області, постановка й формування завдань дослідження	06.05.24 – 10.05.24	
2	Огляд технологій, що використовуються для прогнозування властивостей білків	11.05.24 – 17.05.24	
3	Розробка інформаційної системи з прогнозування стабільності білків з мутаціями	18.05.24 – 20.05.24	
4	Аналіз отриманих результатів	21.05.24 – 28.05.24	
5	Оформлення пояснювальної записки до кваліфікаційної роботи	27.05.24 – 30.05.24	

Здобувач вищої освіти _____ (підпис) Керівник _____ (підпис)

АНОТАЦІЯ

Записка: 53 стор., 15 рис., 1 додаток, 18 джерел.

Обґрунтування актуальності теми роботи – Тема кваліфікаційної роботи є актуальною, оскільки присвячена розв'язанню важливої практичної задачі прогнозування стабільності білків з мутаціями з використанням розробленої інформаційної системи на основі алгоритмів машинного навчання.

Об'єкт дослідження — процес прогнозування біохімічних властивостей білків.

Мета роботи — розробка інформаційної системи прогнозування стабільності білків з одиночними мутаціями на основі алгоритмів машинного навчання.

Методи дослідження — алгоритми машинного навчання.

Результати — розроблено інформаційну систему прогнозування стабільності білків з одиночними мутаціями на основі алгоритмів машинного навчання. Інформаційна система основана на регресійних алгоритмах Random Forest Regessor та Extra Trees Regressor і дозволяє прогнозувати значення стабільності білків на основі експериментальних даних.

ІНФОРМАЦІЙНА СИСТЕМА, АЛГОРИТМИ МАШИННОГО НАВЧАННЯ, ПРОГНОЗУВАННЯ, PYTHON, PANDAS, RANDOM FOREST REGESSOR.

ЗМІСТ

ВСТУП

Актуальність. У сучасному світі біоінженерія займає центральне місце у вивченні та розвитку біологічних систем, що відкриває необмежені можливості для вдосконалення життя людей. Однією з ключових областей досліджень є вивчення властивостей білків з мутаціями, які впливають на їхню структуру та функціональність. Інформаційна система дослідження білків з використанням машинного навчання дасть можливість аналізувати та обробляти величезні обсяги даних швидше та ефективніше, а також прогнозувати можливі результати експериментів на основі попередніх даних.

Об’єкт дослідження. Властивості білків з мутаціями.

Предмет дослідження. Інформаційна система дослідження білків з використанням машинного навчання.

Гіпотеза. Прогнозування стабільності білків з мутаціями можна досягнути використанням алгоритмів машинного навчання

Наукова новизна. На відміну від існуючих аналогів інформаційних систем, описане у даній роботі програмне рішення дозволить спрогнозувати максимально точні значення, представити їх у вигляді діаграм та графіків.

Структура. Дана робота складається зі вступу, аналітичного огляду, постановки задачі дослідження, вибір методики та інструментів для рішення поставленої проблеми, опису програмної реалізації інформаційної системи, висновків, списку використаних джерел.

1 АНАЛІТИЧНИЙ ОГЛЯД

Біомедична інженерія використовує фундаментальні теорії та методи аналізу для застосування в медицині та біології. З розвитком медичних пристроїв і діагностичних експертних систем це стає все більш важливим для покращення охорони здоров'я. Ці технології обробляють складні та багатовимірні дані. Використання в цих системах алгоритмів машинного навчання дозволяє аналізувати сигнали та виявляти захворювання.

Машинне навчання – це галузь інформатики, яка використовує алгоритми для аналізу великих обсягів даних і прогнозування на основі досвіду. Це корисно для різноманітних обчислювальних завдань, таких як фільтрація електронної пошти, розпізнавання шаблонів та багато інших. У біомедичній інженерії машинне навчання використовується для біологічного моделювання та прогнозування.

Передбачення структури білків давно було важливим завданням біохімії, оскільки структура білків визначає їхню функцію. Машинне навчання також використовується в геноміці для вивчення геному та його еволюції. Технології секвенування ДНК на основі машинного навчання використовуються для діагностики спадкових захворювань та інших медичних застосувань.

Машинне навчання також використовується в редагуванні генів і клінічних дослідженнях. Це сприяє розробці ефективних і точних методів діагностики та лікування.

Інші застосування машинного навчання включають прогнозування активності білків, аналіз генетичних мутацій і ранню діагностику захворювань. Однак у біомедичній інженерії важливо враховувати такі характеристики даних, таких як: невеликий обсяг, категорійність даних і їх неоднорідність [1]. Тому,

інтелектуальний аналіз даних і розробка алгоритмів машинного навчання відіграють важливу роль у вирішенні цих завдань.

1.1 Дослідження актуальності проблеми

Розробка інформаційної системи дружньої до користувача потребує сучасних алгоритмів, які набули важливого значення для автоматизації процесів, виявлення закономірностей у великих обсягах даних, передбачення та прийняття рішень на основі великих наборів інформації. На сьогоднішній день машинне навчання охоплює різноманітні класи алгоритмів, які можна розділити на кілька основних категорій, включаючи такі:

1. Навчання з учителем: В цьому типі навчання моделі працюють з відомими даними для передбачення результату або класифікації об'єктів.

2. Навчання без учителя (кластеризація): У цьому випадку моделі аналізують невідмічені дані та групують їх на основі подібності.

3. Напівконтрольоване навчання: Цей тип включає ситуації, де лише деякі дані мають мітки, і моделі намагаються використати цю інформацію для покращення загального навчання.

4. Навчання з підкріпленням: В цьому типі навчання моделі вчаться взаємодіяти з середовищем та максимізують користь від попередніх дій.

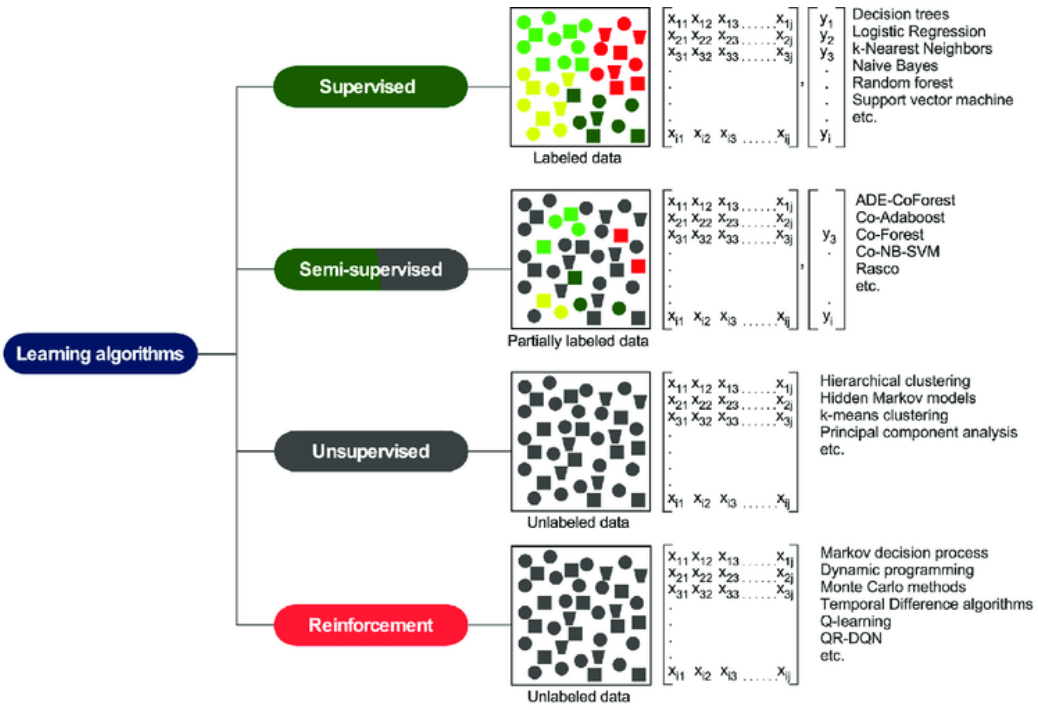


Рисунок 1.1 — Принцип роботи основних класів алгоритмів

Задачі навчання з учителем, такі як класифікація та регресія, є основними напрямками в машинному навчанні. Для їх вирішення використовуються різноманітні алгоритми, такі як лінійна та логістична регресія, дерева рішень, алгоритм опорних векторів (SVM), k-найближчих сусідів (k-NN), наївний баєсовський класифікатор, випадковий ліс та методи зменшення розмірності. Зазвичай для поліпшення результатів використовують ансамблеві методи, такі як беггінг та бустінг.

Беггінг є однією з ключових технік ансамблевого навчання, яка спрямована на підвищення стабільності та точності машинних моделей шляхом зменшення варіативності. Ідея беггінгу полягає у створенні декількох моделей на різних підмножинах даних і об'єднанні їхніх результатів. Це досягається за допомогою методу бутстрепу, де з оригінального тренувального набору даних випадковим чином з поверненням вибираються підмножини, які мають такий самий розмір, як

і початковий набір. Кожна з цих підмножин використовується для тренування окремої моделі, яка зазвичай є слабким учасником (weak learner), таким як дерево рішень. Оскільки різні моделі навчаються на різних наборах даних, вони мають різні помилки. Після того, як всі моделі натреновані, їхні прогнози об'єднуються, наприклад, шляхом усереднення для регресії. Це усереднення знижує дисперсію і, як наслідок, підвищує точність кінцевої моделі, оскільки помилки окремих моделей компенсують одна одну.

Беггінг особливо ефективний для моделей з високою варіативністю, таких як дерева рішень, які схильні до перенавчання. Застосовуючи беггінг, значно зменшується ймовірність перенавчання та покращується узагальнююча здатність моделі. Однією з найвідоміших реалізацій беггінгу є алгоритм Random Forest, який поєднує в собі велику кількість дерев рішень, що навчаються на різних підмножинах даних з різною підмножиною ознак, що ще більше знижує кореляцію між окремими деревами.

З іншого боку, бустінг є іншим підходом ансамблевого навчання, який спрямований на перетворення слабких учасників у сильні за рахунок послідовного навчання моделей. Ключова ідея бустінгу полягає в тому, щоб тренувати моделі послідовно, де кожна нова модель намагається виправити помилки попередньої. На відміну від беггінгу, де моделі навчаються незалежно, в бустінгу кожна наступна модель орієнтується на ті дані, які були неправильно класифіковані або мали найбільшу похибку в попередніх ітераціях.

Один з найпопулярніших алгоритмів бустінгу — це AdaBoost (Adaptive Boosting). У цьому методі кожен зразок даних має вагу, яка визначає його важливість. На початку всі зразки мають рівні ваги. Після тренування першої моделі зразки, які були неправильно класифіковані, отримують збільшені ваги, а ті, що були правильно класифіковані — зменшені. Наступна модель тренується з урахуванням нових ваг, надаючи більше уваги тим зразкам, які були важкими для

попередньої моделі. Цей процес повторюється, і кінцевий прогноз отримується шляхом взваженого голосування всіх моделей.

Іншою відомою реалізацією бустінгу є градієнтний бустінг (Gradient Boosting). У цьому підході кожна нова модель намагається мінімізувати залишкові помилки (residual errors) попередніх моделей, що досягається шляхом оптимізації певної цільової функції за допомогою методів градієнтного спуску. Кожна наступна модель додається до ансамблю, щоб зменшити залишкові похибки, а не просто покращити класифікацію важких зразків, як у випадку з AdaBoost. Градієнтний бустінг має кілька популярних реалізацій, таких як XGBoost, LightGBM та CatBoost, які відомі своєю ефективністю та високою продуктивністю у багатьох змагальних задачах з машинного навчання. Ці алгоритми включають додаткові оптимізації, такі як регуляризація, обтинання дерев, паралельне обчислення, що робить їх надзвичайно потужними для роботи з великими та складними наборами даних.

Нейронні мережі

Нейронна мережа — це модель машинного навчання , яка приймає рішення подібно до людського мозку, використовуючи процеси, які імітують те, як біологічні нейрони працюють разом, щоб ідентифікувати, зважувати варіанти та робити висновки. Кожна нейронна мережа складається з шарів вузлів або штучних нейронів вхідного рівня, одного або кількох прихованих шарів і вихідного рівня. Кожен вузол підключається до інших і має власну вагу та поріг. Якщо вихід будь-якого окремого вузла перевищує вказане порогове значення, цей вузол активується, надсилаючи дані на наступний рівень мережі. В іншому випадку дані не передаються на наступний рівень мережі.

Нейронні мережі покладаються на навчальні дані, щоб навчатися та підвищувати свою точність з часом. Після їх точного налаштування на точність

вони стають потужними інструментами, що дозволяють класифікувати та кластеризувати дані з високою швидкістю. Одним із найвідоміших прикладів нейронної мережі є пошуковий алгоритм Google. Нейронні мережі іноді називають штучними нейронними мережами (ШНМ) або імітованими нейронними мережами (СНМ). Вони є підмножиною машинного навчання та є основою моделей глибокого навчання .

Нейронні мережі можна класифікувати на різні типи, які використовуються для різних цілей. Існують декілька основних різновидів нейромереж.

1. Нейронні мережі прямого зв’язку, або багатошарові персептрони (MLP). Вони складаються з вхідного шару, прихованого шару або шарів і вихідного шару. Дані зазвичай вводяться в ці моделі для їх навчання, і вони є основою для комп’ютерного зору, обробки природної мови та інших нейронних мереж.

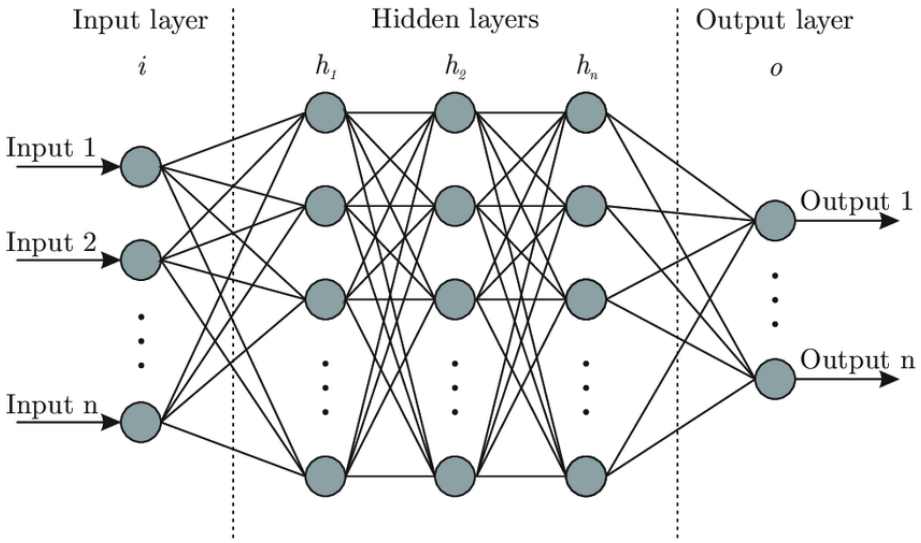


Рисунок 1.2 —Нейронна мережа прямого зв’язку

2. Згорткові нейронні мережі (CNN) схожі на мережі прямого зв’язку, але вони зазвичай використовуються для розпізнавання зображень, розпізнавання

образів або комп’ютерного зору. Ці мережі використовують принципи лінійної алгебри, зокрема множення матриць, щоб ідентифікувати шаблони в зображенні (рис.1.2).

3. Повторювані нейронні мережі (RNN) ідентифікуються за їх петлями зворотного зв’язку. Ці алгоритми навчання в основному використовуються під час використання даних часових рядів для прогнозування майбутніх результатів, наприклад прогнозування фондового ринку чи прогнозування продажів.

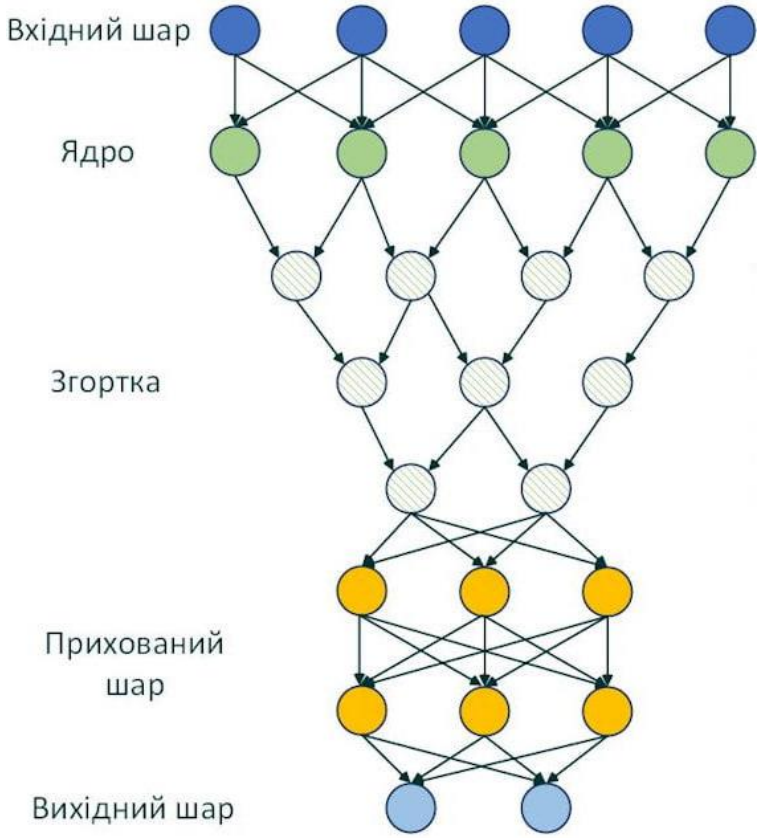


Рисунок 1.3 — Згорткова нейронна мережа

1.2 Аналіз аналогічних проєктів

У зв’язку з тим, що експериментальні дослідження фізико-хімічних властивостей білків є довготривалим і коштовним процесом, популярність алгоритмів з використанням підходів штучного інтелекту постійно зростає. Найпоширенішими з технологій, які здатні прогнозувати структурні та фізико-хімічні властивості є AlphaFold, RoseTTAFold, ProteinNet.

AlphaFold

AlphaFold - це інноваційна технологія, яка прогнозує структури білків за допомогою штучного інтелекту та глибокого навчання [2]. Ця технологія використовується для передбачення та моделювання тривимірної форми білків з високою точністю, що відкриває широкі можливості в біологічних та медичних дослідженнях. AlphaFold працює на базі глибоких нейронних мереж, які були навчені на великому обсязі даних структур білків. Використовуючи ці дані, він аналізує послідовності амінокислот та передбачає їх тривимірну конформацію. Цей підхід дозволяє AlphaFold ефективно вирішувати складні завдання прогнозування білкових структур із високою швидкістю та точністю.

AlphaFold працює, приймаючи амінокислотну послідовність білка як вхідні дані (рис. 1.4). Ця послідовність представлена у вигляді одного ланцюга літер, де кожна літера відповідає конкретній амінокислоті. Система використовує складну нейронну мережу, яка включає кілька основних компонентів. На першому етапі амінокислотна послідовність проходить через енкодер, який перетворює її в представлення, зручне для аналізу. Це представлення враховує не тільки локальні властивості кожної амінокислоти, але й їхні взаємодії з іншими амінокислотами в послідовності. Далі AlphaFold використовує підхід під назвою "contact map prediction", де нейронна мережа передбачає ймовірність контакту між будь-якими

двома амінокислотами в білку. Контактна матриця, що створюється на цьому етапі, є двовимірною і показує, які амінокислоти взаємодіють одна з одною. На завершальному етапі використовується тривимірна згорткова мережа, яка прогнозує просторову конфігурацію білка. Ця мережа допомагає визначити, як амінокислоти розташовані у тривимірному просторі, формуючи остаточну структуру білка.

Одним з ключових елементів успіху AlphaFold є використання потужних методів машинного навчання та великих обсягів даних для тренування моделі. Система використовує бази даних з експериментально визначеними структурами білків, що дозволяє їй навчитися точніше передбачати структури нових білків. Крім того, AlphaFold застосовує техніки, які дозволяють моделі враховувати еволюційні взаємозв'язки між білками, що також сприяє підвищенню точності прогнозів.

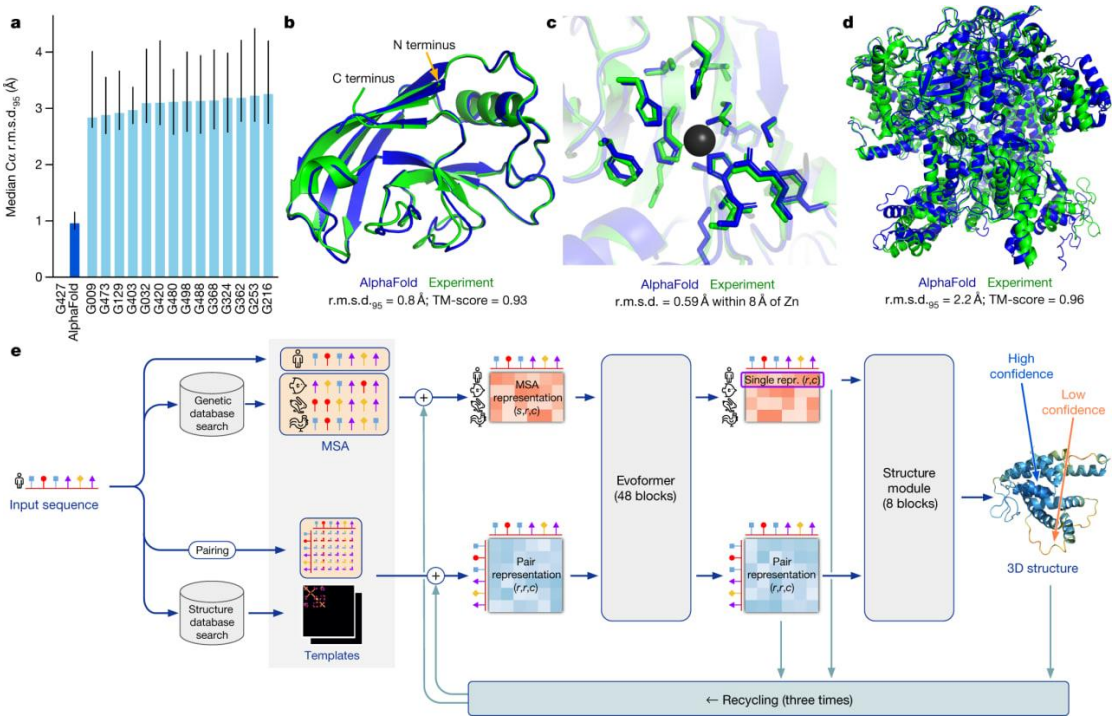


Рисунок 1.4 — Графічна візуалізація роботи AlphaFold

RoseTTAFold

RoseTTAFold використовує технології глибокого навчання, такі як трансформери, для аналізу послідовностей амінокислот та передбачення їх тривимірної структури [3]. Ця система базується на моделі архітектури аналогічній до AlphaFold, але використовує нові підходи та оптимізовані алгоритми для покращення точності та швидкості прогнозування (рис.1.5). Цей проект зробив величезний прорив у прогнозуванні білкових структур шляхом поєднання технології глибокого навчання з традиційними методами прогнозування білків.

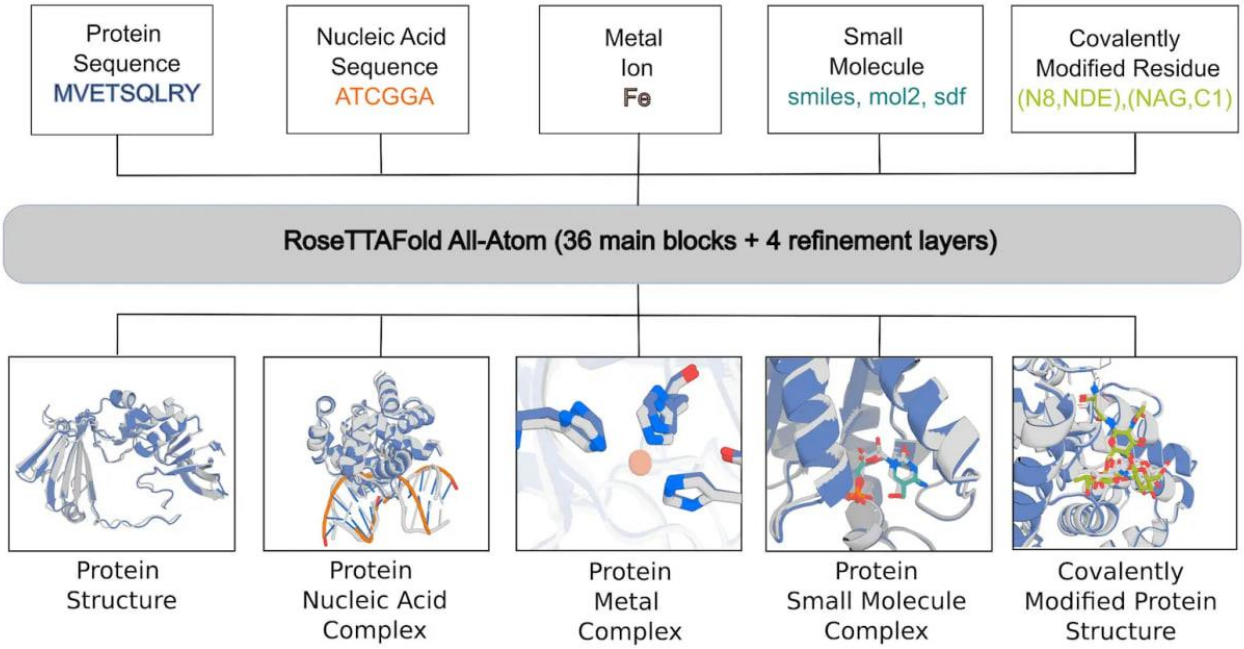


Рисунок 1.5 — Основні напрямки роботи проєкту RoseTTAFold

ProteinNet

ProteinNet - це великий та різноманітний проект у галузі біоінформатики, який використовує методи машинного навчання для аналізу та передбачення

властивостей білків [4]. Цей проект має на меті збільшити нашу розуміння біологічних систем шляхом вивчення структури та функцій білкових молекул. Одним з головних завдань ProteinNet є побудова великих та розносторонніх наборів даних про білки, які включають інформацію про їх амінокислотні послідовності, тривимірні структури, взаємодії з іншими молекулами та біологічні функції. Ці набори даних створюються з використанням різних експериментальних та обчислювальних методів, що дозволяє отримати широкий спектр інформації про білки (рис. 1.6). Одним з ключових компонентів ProteinNet є застосування методів машинного навчання для аналізу цих даних та прогнозування різних властивостей білків [5]. Це включає в себе передбачення тривимірної структури білків, визначення їх функцій та взаємодій з іншими молекулами, а також прогнозування можливих побічних ефектів та властивостей для фармацевтичних досліджень.

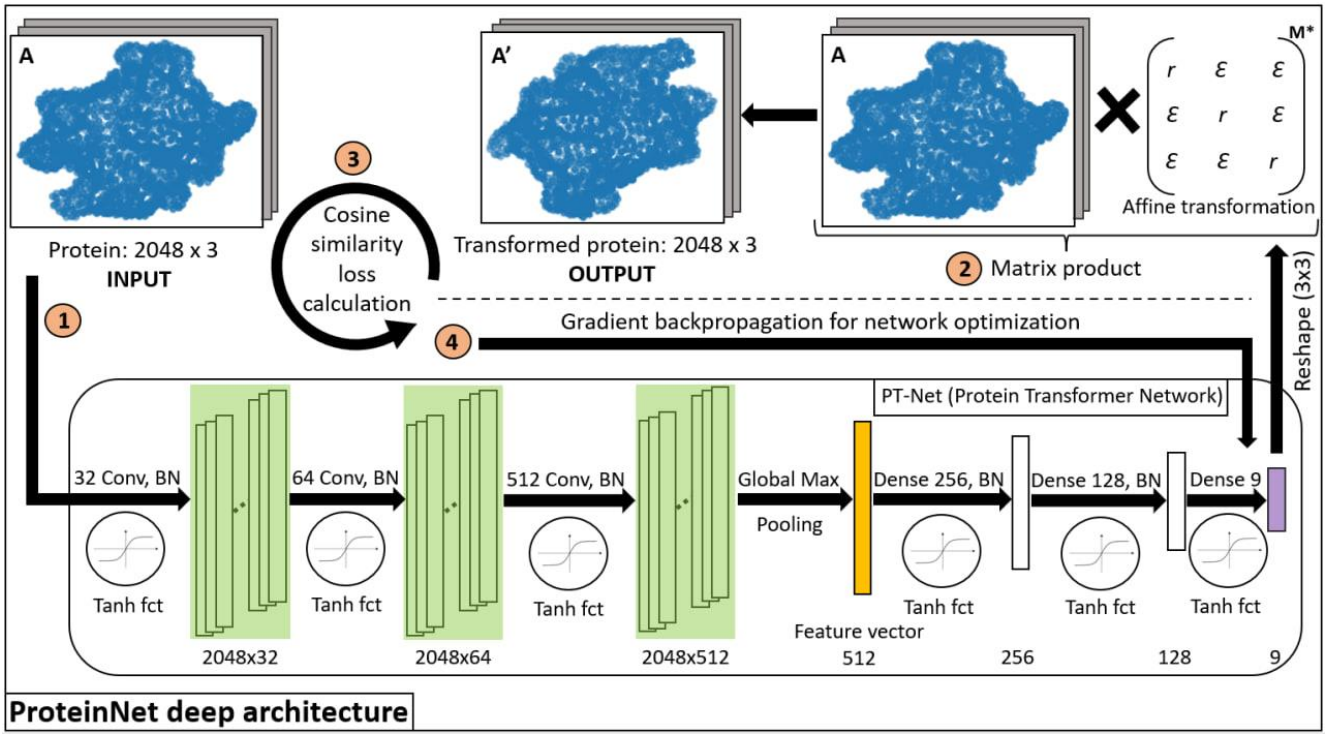


Рисунок 1.6 — Візуалізація архітектури ProteinNet

1.3 Постановка задачі

Метою цієї роботи було створення інформаційної системи машинного навчання для прогнозування стабільності білків на основі фізико-хімічних експериментальних даних. Для досягнення поставленої мети необхідно вирішити наступні задачі:

1. Підготовка, аналіз та обробка вхідних даних, включаючи їх доповнення для створення математичного представлення.

2. Вибір параметрів для навчальної матриці.

3. Вибір і застосування алгоритмів машинного навчання для розробки моделей з метою прогнозування стабільності білків на основі навчальної матриці.

4. Тестування моделі на окремому тестовому наборі даних.

5. Розробка програмної реалізації з використанням мови програмування Python та відповідних бібліотек для обробки та візуалізації даних, а також для реалізації моделей машинного навчання.

6. Подальший аналіз результатів.

Можливості машинного навчання розширюють розуміння та прогнозування взаємозв'язків між параметрами білкових послідовностей. Прогноз стабільності дозволяє створювати нові білкові послідовності з унікальною структурою та корисними властивостями. Аналіз даних, як традиційна складова, є критично важливою, але складною. Одне з основних труднощів у забезпеченні повного аналізу даних для машинного навчання полягає у недостатній кількості числових вхідних даних та обмеженому розмірі наборів даних.

2 ВИБІР МЕТОДІВ РОЗВ'ЯЗАННЯ ЗАДАЧІ

2.1 Вибір програмних засобів

Мова програмування Python

Мова Python має широку спільноту користувачів і розгалужені бібліотеки, що сприяють її поширенню [7]. Мова представлена широким набором бібліотек, спеціально розроблених для задач, пов'язаних з машинним навчанням.

Основні бібліотеки Python включають:

- NumPy
- Pandas
- Matplotlib
- SciPy
- Scikit-Learn
- TensorFlow
- PyTorch

Python відрізняється зрозумілим синтаксисом, що полегшує розробку програм. Він підтримує різні парадигми програмування, такі як об'єктно-орієнтоване, функціональне та імперативне програмування, що забезпечує багато можливостей для створення програм. Python також підтримує розширення для C/C++, що дозволяє використовувати швидкі бібліотеки для оптимізації продуктивності програм [6]. Кросплатформенність - ще одна перевага Python, оскільки код, написаний на цій мові, може працювати на різних операційних системах без змін, що спрощує розгортання програм на різних платформах. Python застосовується в різних сферах, включаючи веб-розробку, наукові дослідження, аналіз даних, штучний інтелект, робототехніку та інші. Це одна з найпопулярніших мов програмування у світі та ідеальний вибір для створення різноманітних програмних продуктів.

Середовище розробки

Для роботи використовувалося інтегроване середовище розробки Anaconda, яке підтримує наукове програмування на мовах Python і R. Важливо відзначити, що Anaconda має відкритий вихідний код та є простим інструментом для створення програмного забезпечення для обробки наукових даних і реалізації завдань машинного навчання [7, 8].

Для роботи з кодом і аналізу результатів використовувався Jupyter Notebook, відкрита веб-програма, яка також включена в пакет Anaconda. Jupyter Notebook дозволяє створювати код, запускати його та миттєво переглядати результати виконання коду. У цьому простому середовищі розробники можуть писати код, додавати анотації та виконувати блоки коду окремо.

Jupyter Notebook - це інтерактивне середовище, яке забезпечує можливість створення та обміну документами, що містять живий код, тексти, візуалізації та пояснення. Використовуючи Jupyter Notebook, ви можете виконувати код по частинах, спостерігаючи за результатами кожного кроку, що робить його ідеальним інструментом для наукових досліджень та аналізу даних. Також, Jupyter Notebook може використовуватися для створення освітніх матеріалів та документації.

2.2 Алгоритми розв’язання задачі

В задачах регресії використовуються різні алгоритми. Було обрано алгоритми регресії Random Forest Regressor, LinearRegression, XGB Regressor, GradientBoostingRegressor з метою обрання найкращого з них [7].

Random Forest Regressor

Random Forest Regressor (RF) - це ефективний метод машинного навчання, який комбінує декілька дерев рішень для вирішення задач класифікації та регресії.

Основна ідея полягає в створенні великої кількості дерев і об'єднанні їх результатів для отримання точного та стабільного прогнозу. RF використовує випадковість на кількох рівнях: вибір випадкових даних для кожного дерева та випадкову підмножину функцій на кожному рівні дерева. Це допомагає уникнути перенавчання та покращити стійкість моделі. Однією з головних переваг RF є можливість обробки даних з різними типами характеристик, включаючи категоріальні та числові. RF також дозволяє ефективно визначати важливість ознак, що полегшує аналіз даних та вибір найбільш важливих для прогнозування. RF широко використовується у різних галузях, включаючи медицину, фінанси та екологію, і може успішно застосовуватися для вирішення проблем великих обсягів даних у класифікації та регресії [8]. Незважаючи на свої переваги, RF має обмеження, такі як збільшений час навчання та менша точність у випадку нечітких або шумних даних. Однак, він залишається важливим інструментом у машинному навчанні завдяки своїй високій точності, стійкості та універсальності.

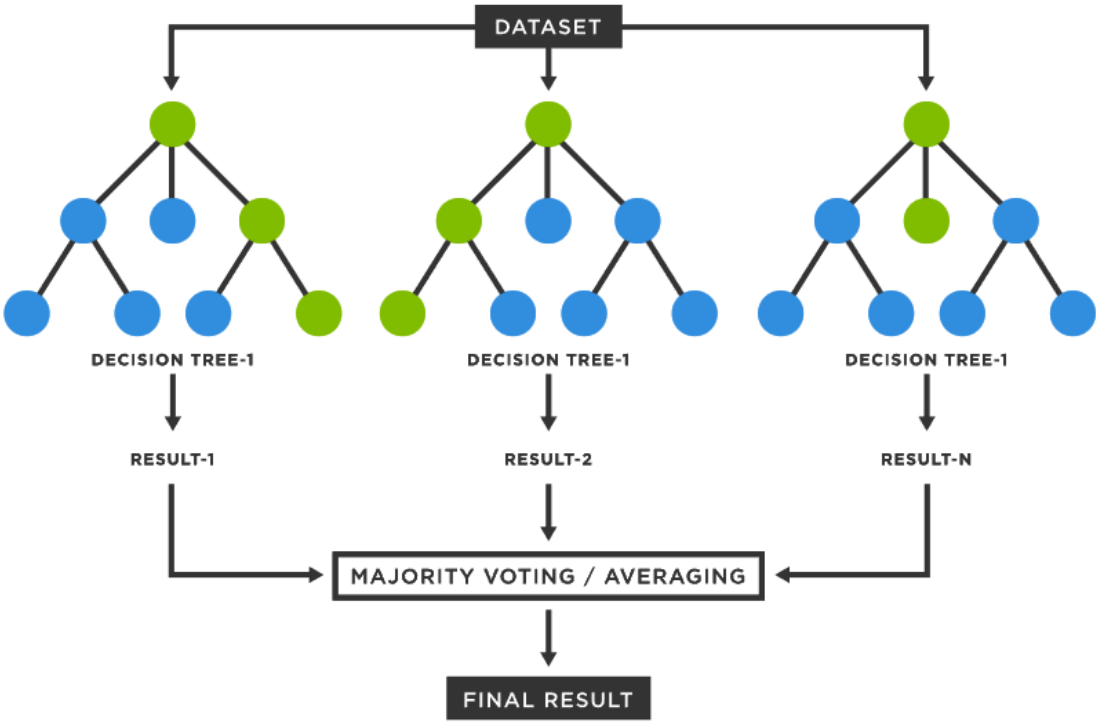


Рисунок 2.1 — Схема алгоритму Random Forest

ExtraTrees

ExtraTrees, або Extremely Random Trees, є модифікацією алгоритму Random Forest (RF), яка пропонує новий підхід до побудови дерев рішень [9, 10]. Замість того, щоб вибирати найкращий параметр для поділу вузла, як це робиться в RF, ExtraTrees випадковим чином вибирає параметри поділу з відповідного діапазону значень для кожного вузла дерева. Це робить алгоритм менш чутливим до вибору оптимальних параметрів і забезпечує певну стійкість моделі до змін у вихідних даних. Однією з ключових переваг ExtraTrees є менший ризик перенавчання. Через випадковість у виборі параметрів поділу, модель має менше ймовірність запам'ятати шаблони в навчальних даних і перетренути. Крім того, завдяки випадковому вибору параметрів, ExtraTrees може працювати швидше, що особливо важливо для великих обсягів даних. Ще одна перевага полягає у продуктивності на багатофункціональних даних [8]. Додаткові дерева можуть ефективно працювати з даними, які містять велику кількість ознак, оскільки випадковість допомагає уникнути перенавчання та забезпечити стійкість до змін у вихідних даних. Узагальнюючи, ExtraTrees є потужним і ефективним алгоритмом машинного навчання, який добре підходить для завдань класифікації та регресії, особливо у випадках, коли необхідно працювати з великими обсягами даних або коли важлива швидкість обробки.

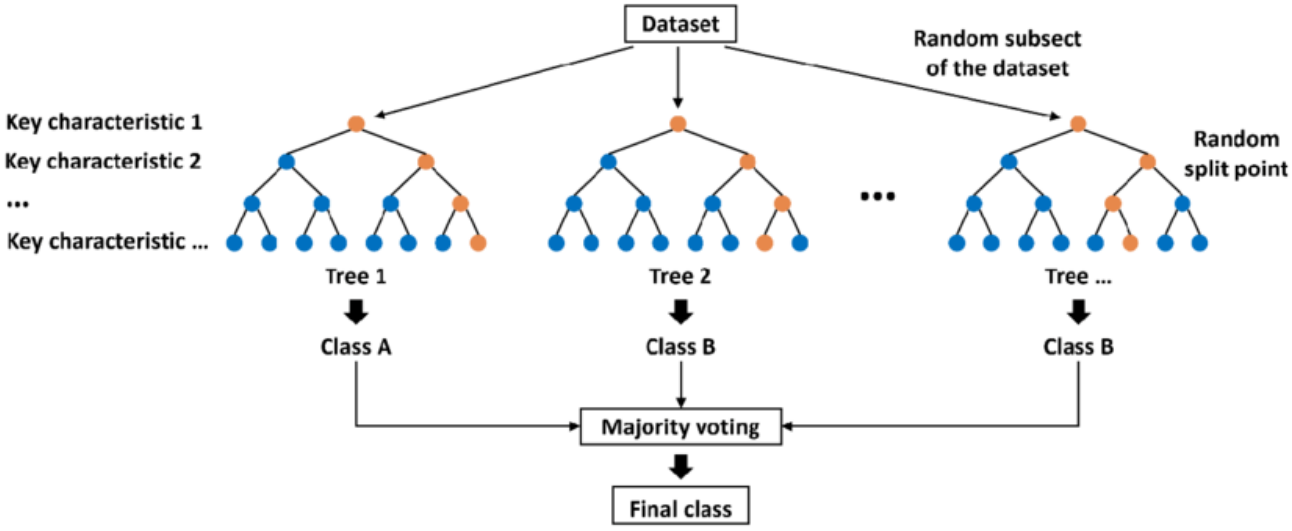


Рисунок 2.2 — Схема алгоритму ExtraTrees

XGB Regressor

XGBoost (Extreme Gradient Boosting) — одна з найпопулярніших і найпотужніших реалізацій градієнтного посилення, розроблена для високої продуктивності та ефективності. XGB Regressor — це спеціальна реалізація XGBoost для проблем регресії, яка використовує набір дерев рішень для побудови надійної моделі з високою потужністю прогнозування.

XGBoost заснований на принципі градієнтного посилення, коли кожна нова модель намагається виправити помилки попередніх моделей. Основна ідея полягає в тому, щоб додати до набору нові дерева рішень, кожне з яких навчатиметься на залишкових помилках попередніх дерев. Це дозволяє системі зосередитися на даних, які важко передбачити, і поступово зменшувати загальну похибку моделі.

XGB Regressor використовує методи другого порядку для оптимізації, що дозволяє краще оцінювати зміни цільової функції та швидше знаходити оптимальні рішення. Ця оптимізація включає не лише розрахунок градієнта, але й функцію Гессе (друга похідна), яка дозволяє точніше та стабільніше оновлювати модель. Важливим

аспектом XGBoost є його здатність обробляти неповні дані та автоматично обробляти відсутні значення, що робить його особливо корисним для реальних програм із неповними наборами даних. Однією з головних переваг XGB Regressor є здатність до регулярізації, яка допомагає уникнути перенавчання моделі. Регуляризація додає штрафні терміни до цільової функції, зменшує вагу складних моделей і сприяє вибору простіших і більш узагальнених моделей. Крім того, Regressor підтримує паралельні та розподілені обчислення, забезпечуючи ефективну обробку великих обсягів даних. Це досягається за допомогою блокової структури дерева, яка дозволяє створювати та оновлювати дерева паралельно. Така архітектура значно збільшує швидкість навчання моделі та скорочує час обробки великих масивів даних. Regressor також включає механізми раннього припинення, які дозволяють припинити навчання, якщо модель не покращується протягом певної кількості ітерацій. Це дозволяє уникнути зайвих обчислень і зменшити ризик перенавчання, зберігаючи продуктивність моделі [9]. Гнучкість алгоритму також відображається в можливості налаштування гіперпараметрів, що дозволяє адаптувати модель до конкретних завдань і даних. Важливі параметри включають кількість дерев, глибину дерева, швидкість навчання та інші, які можна оптимізувати за допомогою перехресної перевірки або інших методів оптимізації гіперпараметрів.

GradientBoostingRegressor

GradientBoostingRegressor є потужним інструментом машинного навчання, який реалізує принцип градієнтного бустінгу для задач регресії. Цей алгоритм поєднує в собі властивості ансамблевих методів, створюючи сильну модель шляхом послідовного додавання слабких учасників, таких як дерева рішень, які навчаються на помилках попередніх моделей. Це дозволяє поступово знижувати загальну похибку та підвищувати точність прогнозів.

Основний принцип роботи GradientBoostingRegressor полягає у побудові послідовності моделей, де кожна нова модель коригує помилки попередньої. Процес починається з простої моделі, яка робить початкові прогнози. Потім залишкові похибки цієї моделі використовуються для навчання наступної моделі [10]. Цей процес повторюється, додаючи нові моделі до ансамблю, поки не досягнеться задана кількість ітерацій або поки похибка не буде зменшена до прийнятного рівня.

2.3 Методи обробки даних

Обробка даних - це ключовий етап у вирішенні будь-якої аналітичної задачі, оскільки від якості підготовки даних залежить ефективність подальшого аналізу та моделювання [11]. Для подальшої роботи з даними було проведено відсіювання та графічне представлення.

Використовуючи стандартне відхилення, можна видалити значення, які перевищують за межами три стандартних відхилень від середнього. Це допоможе зменшити вплив викидів на аналіз та на графічне зображення результатів. Пусті або пропущенні значення (0 та NaN відповідно) можна видалити або заповнити їх середнім або медіанним значенням певного стовпця. Завдяки цьому результати будуть більш реалістичними та узагальненими. Для перевірки виконаних перетворень треба провести статистичний аналіз. Він включає в себе побудову гістограм та тести нормального розподілу. При виконанні тестів нормального розподілу досліджується, наскільки дані у наборі відповідають типовому гаусівському (нормальному) розподілу. Одним з популярних методів є тест Шапіро-Вілка. Цей тест порівнює наші фактичні дані з тим, що очікується від нормального розподілу. Якщо результат менший за певний поріг, ми відкидаємо гіпотезу про те, що дані розподілені нормально. Це може вказувати на те, що вхідні дані мають інший тип розподілу або все ще містять аномальні значення. Ще один

популярний тест - тест Колмогорова-Смірнова. Він також перевіряє, наскільки добре дані відповідають нормальному розподілу, але з іншим підходом до порівняння фактичних даних з очікуваними значеннями [12]. Результати цих тестів допомагають вирішити, чи можливо безпечно застосовувати методи аналізу, що базуються на припущенні про нормальний розподіл.

Якщо розподіл даних не є гаусівським, можна використовувати різні методи перетворення, такі як логарифмічне або квадратне кореневе перетворення. Не гаусівський розподіл може мати різні форми, такі як бімодальний (з двома піковими) або скісний (коли розподіл відхиляється від симетричності). Для ефективної зміни скісного розподілу, використовують логарифмічне перетворення. Його застосовують особливо коли діапазон значень дуже великий. Квадратне кореневе перетворення також може використовуватися для зменшення скосу. Якщо скіс дуже виражений зліва, то ефективно застосувати квадратно кореневе перетворення.

Вибір найбільш важливих ознак для моделювання допомагає покращити ефективність моделі. Це може включати в себе використання методів відбору ознак або бінінг, який дозволяє групувати значення ознак у різні категорії [13]. Бінінг — це процес групування безперервних змінних або числових значень у дискретні інтервали (біни). Цей метод використовується в різних областях, таких як статистика, машинне навчання, аналіз даних, щоб спростити аналіз і зробити дані більш зрозумілими. Однією з ключових причин, чому бінінг важливий, є зменшення шуму. Бінінг допомагає зменшити вплив випадкових коливань або шуму в даних, досягаючи цього шляхом об'єднання значень у більші категорії, що згладжує незначні варіації. Це також покращує інтерпретацію даних. Перетворення безперервних змінних на категорійні може зробити дані більш зрозумілими. Це важлива складова процесу аналізу даних та розробки моделей машинного навчання. Велика кількість ознак може призвести до перенавчання (overfitting)

моделі або збільшення обчислювальних витрат. Тому важливо вибрати лише найбільш інформативні ознаки для використання у моделі.

Крос-валідація (Cross-validation)

Крос-валідація - це метод перевірки продуктивності моделі машинного навчання. Після навчання моделі машинного навчання на маркованих даних, передбачається, що вона працюватиме на нових даних. Однак важливо забезпечити точність прогнозів моделі у виробничому середовищі. Для цього необхідна валідація моделі. Процес валідації включає прийняття рішення про те, чи є числові результати, що кількісно визначають гіпотетичні взаємозв'язки між змінними, прийнятними як описи даних [14]. Щоб оцінити продуктивність моделі машинного навчання, необхідно протестувати її на нових даних. Виходячи з продуктивності моделі на невідомих даних, можна визначити, чи вона недонавчається, переонавчається чи "добре узагальнює".

Однією з технік, що використовується для тестування ефективності моделі машинного навчання, є крос-валідація. Цей метод також є процедурою ресемплінгу, яка дозволяє оцінювати модель навіть за наявності обмежених даних.

Щоб виконати крос-валідацію, частина навчального набору даних відокремлюється від тренінгового набору даних [15, 16, 17]. Ці дані не будуть використані для навчання моделі, але будуть використані пізніше для тестування та валідації моделі.

Крос-валідація часто використовується в машинному навчанні для порівняння різних моделей та вибору найвідповіднішої для конкретної проблеми. Вона є як простою для розуміння, так і легкою для реалізації, а також менш упередженою, ніж інші методи [18]. Основні техніки крос-валідації:

1. Крос-валідація K-fold. Цей метод включає розбиття набору даних на k рівних частин або "фолдів". Модель тренується на k-1 фолдах, а потім тестується на залишковому фолді. Цей процес повторюється k разів, при цьому кожен фолд використовується для тестування один раз. Наприклад, у 5-fold крос-валідації дані діляться на п'ять частин, і модель тренується та тестується п'ять разів.

2. Стратіфікована K-fold крос-валідація. Це варіант K-fold крос-валідації, який зберігає пропорції класів у кожному фолді. Це особливо важливо для незбалансованих наборів даних, де кількість прикладів одного класу значно перевищує інші.

3. Leave-One-Out крос-валідація (LOOCV). Це крайній випадок K-fold крос-валідації, де k дорівнює кількості спостережень у наборі даних. Для кожного спостереження модель тренується на всіх інших спостереженнях і тестується на цьому одному спостереженні. Хоча цей метод дуже точний, він є обчислювально затратним для великих наборів даних

4. Leave-P-Out крос-валідація (LPOCV). Це узагальнення LOOCV, де p спостережень залишаються для тестування, а модель тренується на решті (n-p) спостереженнях. Як і LOOCV, цей метод може бути дуже затратним для великих p або великих наборів даних.

5. Крос-валідація з перемішуванням (Shuffle Split). У цьому методі набір даних випадковим чином перемішується та розбивається на навчальні та тестові набори кілька разів. Це забезпечує більшу варіативність у розподілі даних, що може допомогти краще оцінити узагальнюючу здатність моделі.

6. Time Series крос-валідація. Для часових рядів стандартні методи крос-валідації можуть не підходити через залежність спостережень від часу. У цьому методі дані діляться на послідовні блоки, де модель тренується на минулих блоках

і тестується на майбутніх, зберігаючи хронологічний порядок. Існують і інші різновиди крос-валідації:

7. Bias-Variance Tradeoff. Крос-валідація допомагає в управлінні компромісом між зміщенням і дисперсією. Високе k у K-fold крос-валідації призводить до низького зміщення, але може збільшити дисперсію. Низьке k, навпаки, збільшує зміщення, але зменшує дисперсію.

8. Збалансованість класів. Стратіфікована крос-валідація особливо корисна для наборів даних із незбалансованими класами, щоб забезпечити рівномірний розподіл класів у кожному фолді.

9. Оцінка стабільності моделі. Крос-валідація може надати інформацію про стабільність моделі, тобто як модель варіюється при зміні навчального набору. Це важливо для оцінки надійності моделі в різних сценаріях.

10. Обчислювальна ефективність. Деякі методи крос-валідації, такі як LOOCV, можуть бути обчислювально затратними для великих наборів даних. У таких випадках слід розглядати компроміс між точністю оцінки та обчислювальними витратами.

11. Гіперпараметрична оптимізація. Крос-валідація часто використовується для налаштування гіперпараметрів моделей. Це дозволяє знайти оптимальні параметри, які забезпечують найкращу продуктивність моделі на валідаційних наборах.

12. Використання крос-валідації в ансамблях. Методи ансамблювання, такі як bagging та boosting, можуть використовувати крос-валідацію для поліпшення продуктивності. Наприклад, у методі bagging створюється кілька моделей на різних підмножинах даних, і крос-валідація може бути використана для оцінки кожної з цих моделей.

Загалом крос-валідація є потужним інструментом при роботі з машинним навчанням. Вона дозволяє не лише оцінити продуктивність моделей, але й підвищити їхню надійність та точність. Використання правильних методів крос-валідації допомагає забезпечити, що моделі будуть добре узагальнюватися на нові, невідомі дані, що є ключовим для успішного застосування машинного навчання у поставлених задачах.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Інформаційна модель

В даній роботі розроблена інформаційна система дослідження властивостей білків з використанням алгоритмів машинного навчання. Інформаційна модель системи представлена на рисунку 3.1. В моделі представлено декілька блоків: аналіз та обробка даних, побудова моделі машинного навчання, тренування алгоритму на експериментальних даних та тестування і оцінка моделі на тестових даних.

Рисунок 3.1 — Алгоритм роботи інформаційної моделі

3.2 Програмна реалізація

Вхідний датасет представляв собою набір необробленої інформації, яка містила викиди, нульові та незареєстровані значення. Попередня обробка даних складалася з видалення викидів (тобто значень, які перевищують три стандартних відхилень від середнього), помилкових значень та заповнення пропущених даних. Після попередньої обробки даних із загального набору даних була витягнута підмножина з одиничними мутаціями.

Викиди – це дані, значення яких віддалені від інших, причиною чого можуть бути помилки, отримані в результаті проведення експериментальних досліджень або рідкісні виключні події. Виявлення викидів здійснювалось методом трьох сігм, виходячи з твердження, що для будь-якої випадкової величини вірогідність відхилення від значення математичного очікування є меншою за три середньоквадратичних відхилення. Дотримання правила трьох сігм є необхідною умовою нормального розподілення даних. В даному блоці коду наведено приклад розрахунку значень-викидів значень датафрейму, які потім були видалені.

```
def remove_outliers(dk, col):
    mean = dk[col].mean()
    std = dk[col].std()
    dk_filtered = dk[(dk[col] >= mean - 3 * std) & (dk[col] <= mean + 3 * std)]
    return dk_filtered
```

З метою не втратити велику кількість даних при видаленні, NaN значення були замінені середніми значеннями по кожному стовпцю датафрейма. Це також підвищило загальну чистоту даних і спростило подальшу їх обробку.

```
# Заповнити пропущені значення середнім
def fill_missing_values(dk):
    return dk.fillna(dk.mean())
```

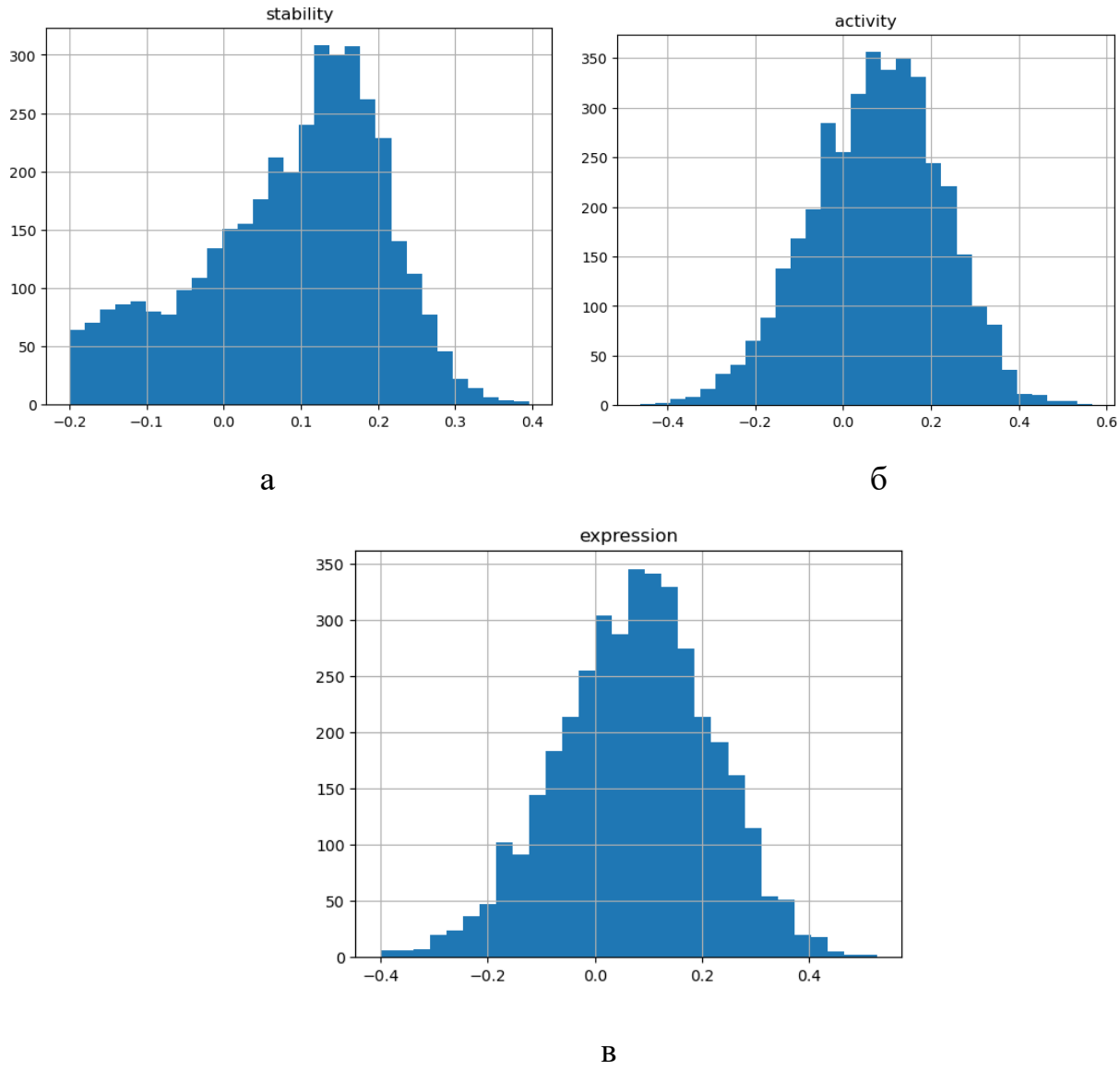
Дисперсійний аналіз підтвердив нормальну поведінку розподілу значень основних показників стабільності S, каталітичної активності A та вираженості (експресії) E у наборі вхідних даних. Тести на нормальний розподіл були проведені

для перевірки того, чи відповідають дані нормальному розподілу, що є одним із базових припущень багатьох статистичних методів. Оцінка здійснювалась за допомогою спеціальних тестів. Надані тести на нормальний розподіл, а саме тест Шапіро-Вілка та Колмогорова-Смірнова. Код, наведений нижче, демонструє програмну реалізацію тестів на нормальний розподіл:

```
# Проведення тестів на нормальний розподіл

def perform_normality_tests(dk):
    normality_results = {}
    for col in dk.columns:
        # Тест Шапіро-Вілка
        shapiro_stat, shapiro_p = shapiro(dk[col])
        # Тест Колмогорова-Смірнова
        ks_stat, ks_p = kstest(df[col], 'norm')
        normality_results[col] = {'Shapiro-Wilk Test': {'statistic': shapiro_stat, 'p-value': shapiro_p}}
                                  'Kolmogorov-Smirnov Test': {'statistic': ks_stat, 'p-value': ks_p}}
    return normality_results

# Виконати обробку даних
processed_data = fill_missing_values(dk)
processed_data = remove_outliers(processed_data, 'stability')

# Побудова гістограм
plot_histograms(processed_data)

# Виконання тестів на нормальний розподіл
normality_results = perform_normality_tests(processed_data)

processed_data
```

Тест Шапіро-Вілка являється тестом на нормальність і перевіряє, чи відповідає вибірка нормальному розподілу. Даний тест корисний для визначення нормального розподілу даних, що є умовою регресії. Тест Колмогорова-Смірнова представляє собою непараметричний метод оцінювання степені відповідності даних заданому розподілу. Наприклад, для значення параметра активності білків значення *p-value* дорівнювало 3.39 (за результатом тесту Шапіро-Вілка) та 2.46 за результатами тесту Колмогорова-Смірнова. З результатів проведення тестів можна зробити висновок, що вхідні дані мають нормальний розподіл, оскільки *p-value*>0.05.

Також були побудовані гістограми розподілення величин. На рисунку 3.2 наведені гістограми розподілу значень для цих параметрів. Симетричний вид підтверджує гіпотезу про нормальний розподіл даних.



а б

в

Рисунок 3.2 — Гістограми розподілення даних навчальної матриці:
а) стабільність, б) активність та в) вираженість

Також в роботі проведено аналіз даних на множинну кореляцію з метою дослідити залежність результативної ознаки від факторних ознак.

За допомогою команди *sns.pairplot()* були побудовані графіки парної залежності параметрів (рис. 3.3). Це дозволило вважати параметр стабільності білків цільовою функцією. З рисунку 3.3 видно, що цільова функція стабільність добре корелює з експресією та активністю, але разом з тим, кореляція з позицією мутації є невираженою. Це пояснюється тим, що стабільність не сильно залежить від позиції, але виключити цей параметр з вхідного датасету не можна за умовою завдання.

```
Ввод [107]: sns.pairplot(ds)
Out[107]: <seaborn.axisgrid.PairGrid at 0x143897150>
```

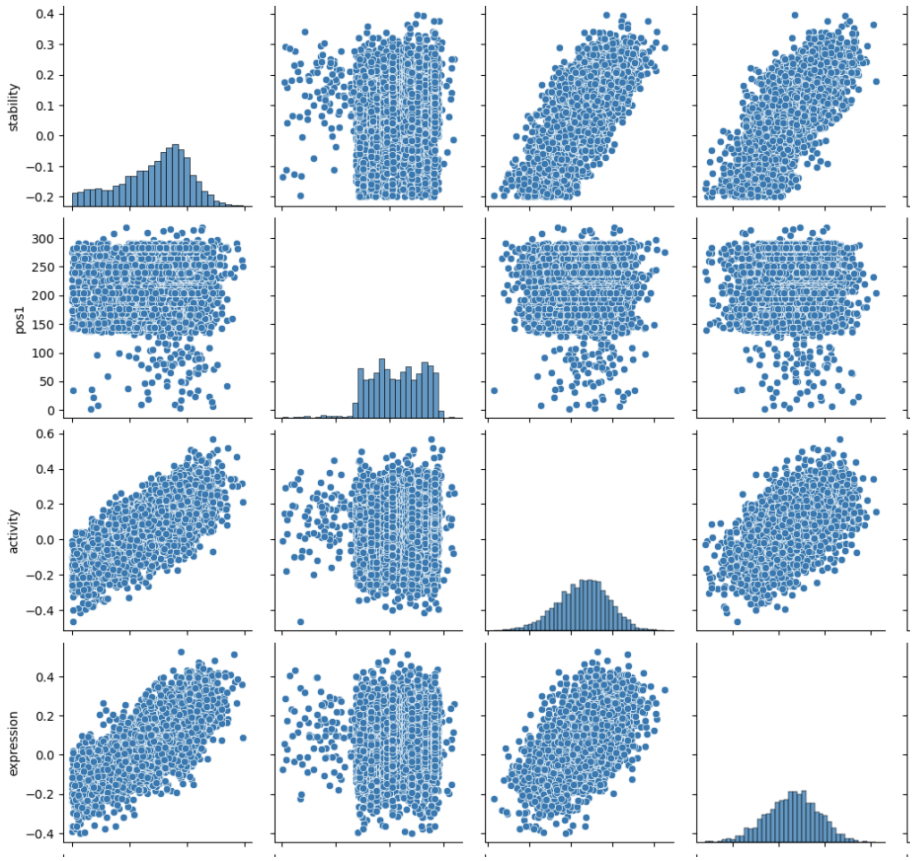


Рисунок 3.3 — Графіки парних відношень значень DataFrame

Оскільки вхідні дані містили дані рядкового типу (категорійні), то було прийнято рішення застосувати бінарне перетворення за допомогою

pd.get_dummies(ds). Мета кодування значень мутованих амінокислот розповсюдженим методом One-Hot – перетворити текстові атрибути в числові значення для подальшої обробки алгоритмами машинного навчання.

Після описаний вище перетворень, вид вхідної матриці мав вигляд, представлений на рисунку 3.4.

	stability	pos1	activity	expression	bulkiness_aa1	polarity_aa1	hydrophobicity_aa1	mutability_aa1	mut1_A	mut1_C	...	mut1_M	mut1_N	mut1_P
0	0.277921	109	0.212063	0.220856	15.77	8.6	-0.7	97.0	False	False	...	False	False	False
1	0.215652	109	0.368016	-0.123210	21.57	5.9	4.2	74.0	False	False	...	False	False	False
2	0.045657	156	0.049663	-0.020267	13.46	5.5	2.5	20.0	False	True	...	False	False	False
3	0.072521	156	0.096945	0.040621	11.68	13.0	-3.5	106.0	False	False	...	False	False	False
4	0.273230	156	0.241501	0.387108	11.68	13.0	-3.5	106.0	False	False	...	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
3848	0.045605	283	0.084889	-0.058486	21.57	5.9	4.2	74.0	False	False	...	False	False	False
3849	0.174018	283	0.310563	0.366504	21.57	5.9	4.2	74.0	False	False	...	False	False	False
3850	0.064218	283	0.093541	0.138007	21.67	5.4	-0.9	18.0	False	False	...	False	False	False
3851	-0.075148	59	-0.199594	-0.075042	13.46	5.5	2.5	20.0	False	True	...	False	False	False
3852	0.100190	59	0.037981	0.031972	13.69	10.4	-3.2	66.0	False	False	...	False	False	False

Рисунок 3.4 — Вид вхідної матриці

Для підбору потрібного алгоритму машинного навчання було застосовано тестування різних моделей за допомогою *pipeline*. Дані з вхідної матриці були поділені на тренінговий та тестовий набір даних у співвідношенні 80 до 20. Були обрані такі регресійні моделі:

- Random Forest,
- LinearRegression,
- XGB Regressor,
- GradientBoostingRegressor.

```
# Split the data into test and train
X_train, X_test, y_train, y_test = train_test_split(
    X,
    y,
    test_size=0.2,
    random_state=0)

models = [
    ('Random Forest', RandomForestRegressor()),
    ('LinearRegression', LinearRegression()),
    ('XGB Regressor', XGBRegressor()),
    ('GradientBoostingRegressor',GradientBoostingRegressor()),
    ]
results = {}
for model_name, model in tqdm_notebook(models):
    # Create a pipeline with a scaler and the current model
    pipeline = Pipeline([
        ('scaler', StandardScaler()),
         ('pca', PCA()),
        (model_name, model)
    ])
    # Fit the model to the training data
    pipeline.fit(X_train, y_train)
    # Make predictions on the testing data
    y_pred = pipeline.predict(X_test)
    y_pred_train = pipeline.predict(X_train)
```

В якості метрик оцінки моделей були обрані коефіцієнт детермінації для тренінгових та тестових даних та середня абсолютна помилка (MAE). Метрика MAE вимірює середню суму абсолютної різниці між фактичним і прогнозованим значеннями:

$$MAE = \frac{1}{n}\sum_{i=1}^{n} Y_i - \widehat{Y}_i,$$

де n – кількість спостережень.

Коефіцієнт детермінації визначався для тренінгового та тестового набору даних за допомогою функції *metrics.r2_score*. Коефіцієнт детермінації - це частка загальної дисперсії змінної, що відображає придатність моделі до змінної, яку вона має намір пояснити. Коефіцієнт детермінації визначається за формулою:

$$R^2 = \frac{\sum_{t=1}^{T}(\hat{Y}_t - \bar{Y})^2}{\sum_{t=1}^{T}(Y_t - \bar{Y})^2},$$

що описується як співвідношення суми квадратів відхилень обумовлених регресією ($\sum_{t=1}^{T}(\hat{Y}_t - \bar{Y})^2$) та залишкової суми квадратів ($\sum_{t=1}^{T}(Y_t - \bar{Y})^2$), яка не пояснюється регресією і характеризує вплив на пояснювальну змінну всіх неврахованих чинників.

```
# Evaluate the model
accuracy_test = metrics.r2_score(y_test, y_pred)
accuracy_train = metrics.r2_score (y_train, y_pred_train)
mae = np.mean(abs(y_pred - y_test))
```

Результати оцінки моделей наведені на рисунку 3.5:

```
Model: Random Forest
R2Train: 0.9517
R2Test: 0.6740
MAE: 0.0552
========================================
Model: LinearRegression
R2Train: 0.6844
R2Test: 0.6827
MAE: 0.0547
========================================
Model: XGB Regressor
R2Train: 0.7059
R2Test: 0.5693
MAE: 0.0639
========================================
Model: GradientBoostingRegressor
R2Train: 0.7542
R2Test: 0.7028
MAE: 0.0530
========================================
```

Рисунок 3.5 — Оцінка якості алгоритмів машинного навчання

В результаті тестування різних моделей машинного навчання видно, що для моделі Random Forest значення коефіцієнта детермінації на тренінговому наборі даних складало 95%, а для тестового – 67% при MAE = 0.06, що показує гарний

результат якості моделі. Наприклад, у випадку лінійної регресії значення коефіцієнта детермінації для тестового набору даних було незначно вище, натомість для тренінгового значно нижче і складало лише 68%. Алгоритм XGB Regressor показав найнижчу якість моделі, r^2 тестового набору даних дорівнював 0.57. Алгоритм GradientBoosting показав гарні результати на тестовому наборі даних – 0.70, на тренінговому – 0.75.

Після проходження тестів був обраний основний алгоритм машинного навчання для обробки даних Random Forest Regresor.

Були проаналізовані результати прогнозування стабільності білків за допомогою розробленої інформаційної системи прогнозування. Для цього був проведений кореляційний аналіз прогнозований даних та вхідних експериментальних з тестового набору даних. Для аналізу використовувався коефіцієнт Пірсона, який вимірює лінійну залежність між двома наборами даних.

Кореляція методом Пірсона між реальними та прогнозованими даними для тестового набору даних складала 97% (`dd_show.corr(method='pearson')`).

Результат співвідношення експериментальних даних з тестового набору даних до прогнозованих відображено на рисутнку 3.6. Результати говорять про гарну відповідність обраної моделі.

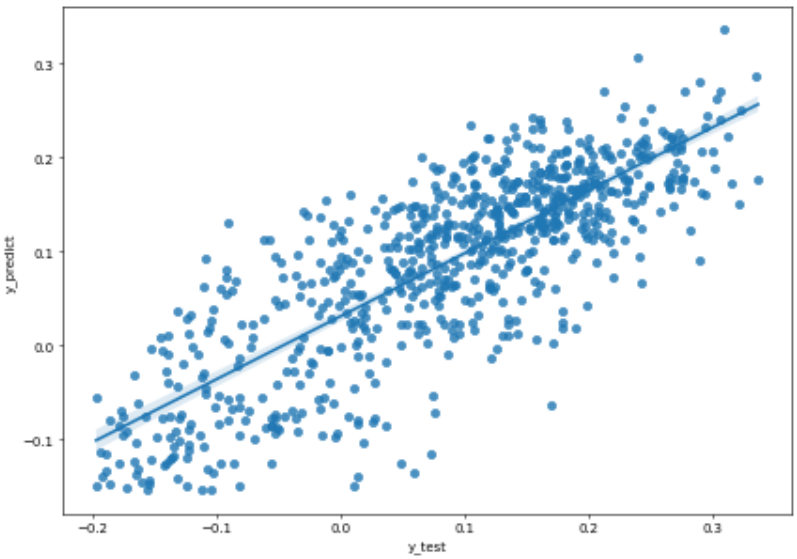


Рисунок 3.6 — Кореляція експериментальних та прогнозованих даних з тестового набору даних

В результаті аналізу отриманих результатів були побудовані навчальні криві (рис.3.7), з рисунку видно, що для моделі достатньо мінімального набору даних приблизно в 400 рядків для досягнення задовільної якості прогнозованих даних.

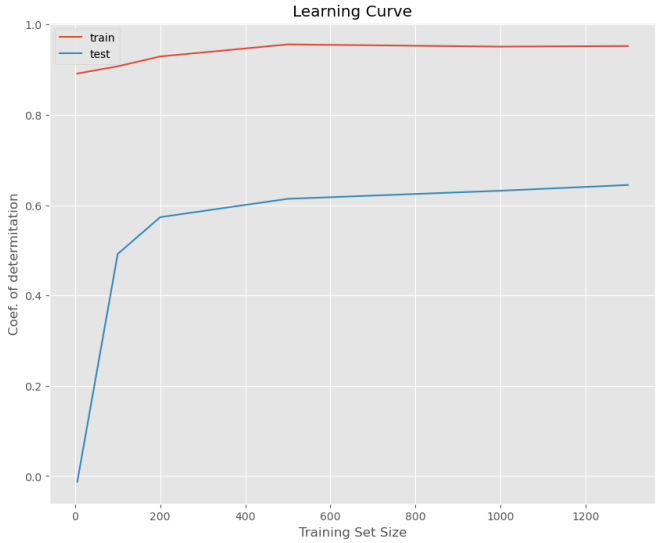


Рисунок 3.7 — Вид навчальних кривих

З метою підвищити якість моделі біла спроба застосувати технологію беггінг `bdt = BaggingRegressor(RandomForestRegressor()).fit(X_train, y_train)`, але очікуваного результату це не дало, точність прогнозування даних в результаті імплементації беггінгу не змінилась.

ВИСНОВКИ

В дипломній роботі було проведено аналіз наукових джерел та алгоритмів машинного навчання, які використовуються в галузі біоінженерії в інформаційних технологіях. Також створено огляд доступних програмних рішень для вирішення поставленої задачі. На основі аналітичного огляду сучасних технологій розроблено інформаційну систему дослідження властивостей білків з використанням машинного навчання. Було проведено аналіз та підготовку даних, обрано та впроваджено алгоритм машинного навчання, проведений аналіз отриманих даних. Інформаційна система вирішує проблему прогнозування параметра стабільності білків та побудована з використанням мови програмування Python та бібліотек Pandas, scipy, sklearn, numpy, Mathplotlib, Seaborn.

Отримані результати відповідають відомим літературним даним, що говорить про високу якість запропонованої інформаційної моделі машинного навчання.

Розроблена інформаційна система, яка використовує методи машинного навчання, може бути застосована для класу подібних задач прогнозування у сфері біоінженерії.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Hernández I.M., Dehouck Y., Bastolla U., López-Blanco J.R., Chacón P. Predicting protein stability changes upon mutation using a simple orientational potential. Bioinformatics. 2023 Jan 1;39(1):btad011.

2. McBride J.M., Polev K., Abdirasulov A., Reinharz V., Grzybowski B.A., Tlusty T. AlphaFold2 can predict single-mutation effects. Phys Rev Lett. 2023 Nov 24; 131(21):218401.

3. KrishnaK. et al. Generalized biomolecular modeling and design with RoseTTAFold All-Atom.Science384, eadl 2528 (2024).

4. AlQuraishi M. ProteinNet: a standardized data set for machine learning of protein structure. BMC Bioinformatics 20, 311 (2019).

5. Bishop Ch. Pattern Recognition and Machine Learning. – Springer, 2016. – 738 p.

6. Swamynathan M. Mastering Maching Learning with Python in Six Steps. A Practical Implementation Guide to Predictive Data Analytics Using Python. – Karnataka: Apress, 2017. – 374p.

7. Bowles M. Machine Learning in Python. – Indianapolis: John Wiley & Sons, 2017. - 361 p.

8. Richert W., Coelho L. Building Machine Learning Systems with Python. – Birmingham: Packt Publishing, 2018. – 326 p.

9. Hastie T., Tibshirani R., Friedman J. The Elements of Statistical Learning Data Mining, Inference, and Prediction. - Springer. – 2019. - 765 p.

10. McKinney W. Python for Data Analysis. – Sebastopol: O’Reilly Media, 2019. – 470 p.

11. Nelli F. Python Data Analytics, Data Analysis and Science Using Pandas, matplotlib, and the Python Programming Language. – New York: Springer Science+Business Media New York, 2018 - 370 p.

12. Marskand S. Machine Learning: An Algorithmic Perspective. – New York: Taylor & Francis Group, 2019. – 452 p.

13. Geron A. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. - Sebastopol: O'Reilly Media, 2019. -574 p.

14. Bowles M. Machine Learning in Python. – Indianapolis: John Wiley & Sons, 2017. - 361 p.

15. Unpingco J. Python for Probability, Statistics, and Machine Learning. -San Diego: Springer, 2018. – 288 p.

16. Duchesnay E., LöfstedtT. Statistics and Machine Learning in Python. – New York: Springer, 2017. – 169 p.

17. Swamynathan M. Mastering Maching Learning with Python in Six Steps. A Practical Implementation Guide to Predictive Data Analytics Using Python. – Karnataka: Apress, 2017. – 374p.

18. Goodfellow I., Bengio Y., Courville A. Deep Learning. - New York: The MIT Press. 2017. – 801 p.

ДОДАТОК

```
from sklearn.preprocessing import MinMaxScaler, StandardScaler
import sklearn.metrics as metrics
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from numpy.polynomial.polynomial import polyfit
from sklearn import linear_model
import pandas as pd
from scipy.stats import shapiro
from tqdm.notebook import tqdm_notebook
import matplotlib.pyplot as plt
from sklearn.model_selection import cross_val_score
get_ipython().run_line_magic('matplotlib', 'inline')
import math
import numpy as np
import seaborn as sns
from scipy import stats
import numpy as np
from mlxtend.plotting import plot_learning_curves

RANDOM_SEED = 42
np.random.seed(RANDOM_SEED)

import sys
if not sys.warnoptions:
    import warnings
```

```
# Генеруємо activity як лінійну функцію від stability з додаванням шуму
b = 0.9
activity_noise = np.random.normal(0, 0.1, num_samples)  # менше шуму для кращої кореляції
ds['activity'] = b * ds['stability'] + activity_noise

# Генеруємо expression як лінійну функцію від stability з додаванням шуму
a = 0.85
expression_noise = np.random.normal(0, 0.1, num_samples)  # менше шуму для кращої кореляції
ds['expression'] = a * ds['stability'] + expression_noise
ds


sns.pairplot(ds)


# Plot histograms for stability, activity, and expression
hist_mean = ds.hist(column='stability', bins=30)
hist_mean = ds.hist(column='activity', bins=30)
hist_mean = ds.hist(column='expression', bins=30)

# Assuming ds is another DataFrame
ds['mut1'] = ds['aa_mutation_syn'].apply(lambda x: x[-2:])
ds['wt1'] = ds['aa_mutation_syn'].apply(lambda x: x[:1])
ds['pos1'] = ds['aa_mutation_syn'].apply(lambda x: x[1:-2]).astype(int)
ds
```

```
ds= ds[['stability', 'mut1', 'wt1', 'pos1', 'activity','expression']]#
ds['mut1'] = ds['mut1'].str.strip()
#ds['aa1']=ds['wt1'] + ds['pos1'].astype(str) + ds['mut1']

ds=ds.dropna()
ds.reset_index(drop=True, inplace=True)
ds

dk = dk[['activity', 'expression','stability']]

def remove_outliers(dk, col):
    mean = dk[col].mean()
    std = dk[col].std()
    dk_filtered = dk[(dk[col] >= mean - 3 * std) & (dk[col] <= mean + 3 * std)]
    return dk_filtered

# Заповнити пропущені значення середнім або медіанним
def fill_missing_values(dk):
    return dk.fillna(dk.mean())

# Побудова гістограм для кожного стовпця
def plot_histograms(dk):
    for col in dk.columns:
        sns.histplot(dk[col], kde=True)
        plt.title(f'Histogram of {col}')
        plt.xlabel(col)
        plt.ylabel('Frequency')
```

```
  plt.show()

# Проведення тестів на нормальний розподіл
def perform_normality_tests(dk):
    normality_results = {}
    for col in dk.columns:
        # Тест Шапіро-Вілка
        shapiro_stat, shapiro_p = shapiro(dk[col])
        # Тест Колмогорова-Смірнова
       ks_stat, ks_p = kstest(df[col], 'norm')
        normality_results[col] = {'Shapiro-Wilk Test': {'statistic': shapiro_stat, 'p-value':
shapiro_p},
       'Kolmogorov-Smirnov Test': {'statistic': ks_stat, 'p-value': ks_p}}
    return normality_results

# Виконати обробку даних
processed_data = fill_missing_values(dk)
processed_data = remove_outliers(processed_data, 'stability')

# Побудова гістограм
plot_histograms(processed_data)

# Виконання тестів на нормальний розподіл
normality_results = perform_normality_tests(processed_data)

processed_data
```

```
# Adding parameters
#table data
dn = pd.read_csv("table_data.csv", on_bad_lines='skip',  delimiter = ",", header=0,
low_memory=False, encoding ='unicode_escape', names = ('AA','Bulkiness','Polarity',
'Hydrophobicity', 'Mutability', 'Hydrop', 'Polar','Hydro','Pol',
'Charge','Hydrophobicity1'))   #    encoding = 'utf-8',
dn['AA'] = dn['AA'].str.strip()
dn

for i in tqdm_notebook(ds.index):
        for j in dn.index:
            if ds.at[i,'mut1'] == dn.at[j,'AA']:
                ds.at[i,'bulkiness_aa1'] = dn.at[j,'Bulkiness']
                ds.at[i,'polarity_aa1'] = dn.at[j,'Polarity']
                ds.at[i,'hydrophobicity_aa1'] = dn.at[j,'Hydrophobicity1']
                ds.at[i,'mutability_aa1'] = dn.at[j,'Mutability']

ds.dropna()

sns.pairplot(ds)
dd_bin = pd.get_dummies(ds)
dd_bin =dd_bin.dropna()
dd_bin
# RandomForestParam

from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
```

```
#plt.scatter(y_test, y_pred, s = 80)
sns.regplot(x="y_test", y="y_predict", data=dd_show);
# Bagging
import numpy as np
from sklearn.metrics import r2_score
get_ipython().run_line_magic('matplotlib', 'inline')
from matplotlib import pyplot as plt
plt.style.use('ggplot')
plt.rcParams['figure.figsize'] = 10, 6
import seaborn as sns
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import  BaggingRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.datasets import make_circles
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(dd_bin.drop(['epistasis'],axis=1),
dd_bin['epistasis'], random_state=42)
# One decision tree regressor
dtree = DecisionTreeRegressor().fit(X_train, y_train)
d_predict = dtree.predict(X_test)
print("Accuracy on test dataset
DecisionTreeRegressor: {:.2f}".format(r2_score(y_test,d_predict)))

# Bagging decision tree regressor
bdt = BaggingRegressor(DecisionTreeRegressor()).fit(X_train, y_train)
bdt_predict = bdt.predict(X_test)
bdt_train_predict = bdt.predict(X_train)
```

```
print(r2_score(y_test,bdt_predict))
print("Accuracy on test dataset
BaggingRegressor:{:.2f}".format(r2_score(y_test,bdt_predict)))
print("Accuracy on train dataset:{:.2f}".format(r2_score(y_train,bdt_train_predict)))
print(np.mean(abs(bdt_predict - y_test)))
# Random Forest
rf = RandomForestRegressor(n_estimators=10).fit(X_train, y_train)
rf_predict = rf.predict(X_test)
rf_train_predict = rf.predict(X_train)
print("Accuracy on test dataset
BaggingRegressor:{:.2f}".format(r2_score(y_test,rf_predict)))
print("Accuracy on train dataset:{:.2f}".format(r2_score(y_train,rf_train_predict)))
dd_show = pd.DataFrame()
dd_show['y_test'] = y_test
dd_show['y_predict'] = bdt_predict
print(dd_show.corr(method='pearson'))
print(dd_show.corr(method='spearman'))
from sklearn.model_selection import learning_curve
import matplotlib.pyplot as plt

X = dd_bin.drop(['epistasis'],axis=1)
y = dd_bin['epistasis']
# overfitting
rand = RandomForestRegressor()
train_sizes, train_scores, test_scores = learning_curve(
    estimator=rand,
    X=X,
```

```
y=y,
    cv=10,
    scoring="r2",#neg_mean_absolute_error",#",
    train_sizes = [ 5, 100, 200, 500, 1000, 1300]
)
train_mean = train_scores.mean(axis=1)# -train_scores.mean(axis=1)
test_mean = test_scores.mean(axis=1)#   -test_scores.mean(axis=1)

plt.subplots(figsize=(10,8))
plt.plot(train_sizes, train_mean, label="train")
plt.plot(train_sizes, test_mean, label="test")

plt.title("Learning Curve")
plt.xlabel("Training Set Size")
plt.ylabel("Coef. of determitation")
plt.legend(loc="best")
plt.show()

# Loading some example data
X_train, X_test, y_train, y_test = train_test_split(dd_bin.drop(['epistasis'],axis=1),
dd_bin['epistasis'], random_state=15)
clf = RandomForestRegressor(n_estimators = 100)

plot_learning_curves(X_train, y_train, X_test, y_test, clf)
#plt.ylabel("Coef. of determitation")
plt.show()
```

```
#Функція для побудови кривих навчання
def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None, n_jobs=None,
train_sizes=np.linspace(.1, 1.0, 5)):
    plt.figure()
    plt.title(title)
    if ylim is not None:
        plt.ylim(*ylim)
    plt.xlabel("Training examples")
    plt.ylabel("Score")
    train_sizes, train_scores, test_scores = learning_curve(
        estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    plt.grid()

    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.1,
                     color="r")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1, color="g")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
             label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
             label="Cross-validation score")
```

```
plt.legend(loc="best")
    return plt

# Побудова кривих навчання для Random Forest
title = "Learning Curves (Random Forest)"
cv = 5  # Кількість перехресних перевірок
plot_learning_curve(rf_random_search.best_estimator_, title, X_train, y_train, cv=cv)

# Крос-валідація для Random Forest
rf_scores = cross_val_score(rf_random_search.best_estimator_, X_train, y_train,
cv=cv)
print("Cross-validated R^2 scores for Random Forest:", rf_scores)

# Побудова кривих навчання для Extra Trees
title = "Learning Curves (Extra Trees)"
plot_learning_curve(et_random_search.best_estimator_, title, X_train, y_train, cv=cv)

# Крос-валідація для Extra Trees
et_scores = cross_val_score(et_random_search.best_estimator_, X_train, y_train,
cv=cv)
print("Cross-validated R^2 scores for Extra Trees:", et_scores)

plt.show()
```