МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
Центр заочної, дистанційної та вечірньої форм навчання

Кафедра комп'ютерних наук

Кваліфікаційна робота магістра

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ МАШИННОГО НАВЧАННЯ В ЗАДАЧАХ БІОІНЖЕНЕРІЇ

Здобувач освіти гр. ІН.мз–11с Олена ПРОЦЕНКО

Науковий керівник, Надія ТИРКУСОВА
доцент, кандидат фізико-математичних наук

В. о. завідувача кафедри Ігор ШЕЛЕХОВ
доцент, кандидат технічних наук

Суми 2022

Сумський державний університет
(назва вузу)

Факультет *ЦЗДВФН* Кафедра *Комп ́ютерних наук*

Спеціальність *«Комп'ютерні науки»*

Затверджую:
В.о.зав.кафедри _____
_____
"_____" _____2022_р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТОВІ

Проценко Олені Борисівні
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) *Інформаційна технологія машинного навчання в задачах біоінженерії*

затверджую наказом по інституту від "____" _____ 20 ___ р. № _____

2. Термін здачі студентом закінченого проекту (роботи)
_____

3. Вхідні данні до проекту (роботи) _____
_____
_____
_____
___

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
1)Інформаційний огляд; 2)Вибір програмних засобів; 3)Практична реалізація. Розробка інформаційної технології машинного навчання; 4)Висновки

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 _____

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів дипломного проекту (роботи)	Термін виконання проекту (роботи)	Примітка
1.	Аналіз проблеми. Інформаційний огляд. Постановка задачі дослідження		
2.	Огляд алгоритмів машинного навчання		
3.	Алгоритми розв’язання задачі регресії		
4.	Розробка інформаційної технології машинного навчання		
5.	Оформлення пояснювальної записки до кваліфікаційної магістерської роботи		

Студент – дипломник _____
(підпис)

Керівник проекту _____
(підпис)

РЕФЕРАТ

Записка: 64 стр., 23 рис., 1 таблиця 1 додаток, 19 літературних джерел.

Об’єкт дослідження — інформаційна технологія машинного навчання в задачах біоінженерії.

Мета роботи — розробка та програмна реалізація системи для прогнозування властивостей біологічних матеріалів з використанням інформаційної технології на основі алгоритмів машинного навчання.

Результати — проведено аналіз сучасних джерел, методів та алгоритмів інформаційної технології в задачах біоінженерії. Розроблено та програмно реалізовано інформаційну технологію машинного навчання для прогнозування властивостей біологічних матеріалів з використанням інформаційної технології на основі регресійних алгоритмів машинного навчання з використанням Python та бібліотек Pandas, TensorFlow та інших.

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ МАШИННОГО НАВЧАННЯ, АЛГОРИТМ, РЕГРЕСІЙНИЙ АНАЛІЗ, PYTHON, PANDAS, TENSORFLOW, БІОІНЖЕНЕРІЯ, ГЛИБОКЕ НАВЧАННЯ

ЗМІСТ

ВСТУП

Останніми роками збір і аналіз біомедичних даних у поєднанні з досягненнями в галузі штучного інтелекту дозволили досягти прогресу у біоінженерії. У цьому контексті машинне навчання показало, що воно є дуже перспективним інструментом як для отримання нових ідей на основі даних, пов’язаних із медициною, фармакологією, так і для досягнення різних цілей і пошуку нових альтернативних рішень, які потенційно можуть бути застосовані для вирішення складних біомедичних проблем у більш автоматизованому та сприятливому режимі. Пошук алгоритмів і нових підходів для проведення досліджень, застосування стратегій машинного навчання в біомедичній галузі, включаючи обробку зображень і сигналів, спрямовану на діагностику та реабілітацію, є постійним процесом. Не менш важливою є і задача регресії для обробки експериментальних даних. При цьому, алгоритми машинного навчання прискорюють біомедичні дослідження і надають можливість проектувати нові можливі біологічні об’єкти.

Методи машинного навчання вже знайшли широке застосування у генетиці та геноміці. Вони виявилися найбільш корисними для інтерпретації великих наборів геномних даних та анотації великого числа елементів геному. Методи машинного навчання були успішно застосовані для розпізнавання стартової транскрипції, альтернативного сплайсингу, промотерів, енхансерів, розташування нуклеосом Тому, задачею даної роботи була розробка інформаційної технології машинного навчання для задач, що пов’язані з експериментальними даними в галузі біоінженерії з метою прогнозування їх властивостей для розробки новітніх біозразків з заданими властивостями.

1 ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1. Використання інформаційної технології машинного навчання в біоінженерії

Машинне навчання можна визначити як галузь інформатики, яка використовує алгоритми для вивчення великої кількості даних і прогнозування на основі експериментального досвіду. Машинне навчання застосовується до широкого діапазону обчислювальних завдань, таких як фільтрація електронної пошти, оптичне розпізнавання образів, комп'ютерний зір. У багатьох таких областях, де проектування явні алгоритми з хорошою продуктивністю складні або нездійсненно, машинне навчання є найефективнішою технологією [1].

В галузі біомедичної інженерії алгоритми машинного навчання застосовуються в багатьох задачах, наприклад, в задачі біологічного моделювання. Машинне навчання дозволяє використовувати алгоритми, спроможні навчатися на експериментальному досвіді, щоб мати можливість робити майбутні прогнози [2].

Біомедична інженерія — це концепція застосування фундаментальних теорій і аналітичних практик до медицини та біології. Це може бути ефективним у сфері охорони здоров'я від впровадження медичних приладів до діагностичних експертних систем. Такі пристрої та експертні системи продукують багатовимірні та нерегулярні дані. Використання алгоритмів машинного навчання для обробки сигналів цих пристроїв є ефективним для аналізу даних та ідентифікації захворювань. Застосування машинного навчання у біомедичній інженерії можна представити як аналіз біо- та медичних зображень, секвенування генома та аналіз експресії генів.

Вивчення властивостей з метою покращення властивостей протеїну є дуже популярним напрямком у біоінженерії, біології, біотехнології та медицині. Прогнозування структури білка довгий час було центральною

проблемою в біохімії, яка базувалась на тому, що структура молекули визначає її функції.

Застосування машинного навчання в біології та біоінформатиці дуже важливе і ефективне. Найчастіше використовується у геноміці, яка зосереджена на вивченні картографування геному, еволюції та редагування. Геном — це повний набір генетичного матеріалу, що міститься в організмі.

Також методи машинного навчання знайшли використання у секвенуванні геному, що наразі відіграє ключову роль у медичній діагностиці. Технології секвенування ДНК на основі машинного навчання використовують для діагностики спадкових захворювань, встановлення батьківства, клонування генів, виділення нових генів [3].

Машинне навчання застосовується у редагуванні генів при маніпулюванні генетичним складом організму шляхом вставки, видалення або заміни послідовності ДНК. Проте дослідники постійно працюють над вибором правильної послідовності ДНК з заданими властивостями, що являється тривалим процесом, схильним до помилок. Машинне навчання прийшло на допомогу, полегшивши ідентифікацію даних, значно скоротивши вартість і час, необхідні для редагування генів.

Машинне навчання широко використовується в клінічному процесі, наприклад, для доступу до даних пацієнтів, які містяться в електронних записах, паперових картах та інших джерелах.

Стосовно генної інженерії, відомо багато наукових статей, які описують різні підходи до прогнозування білкової придатності в аналізі генів з метою визначення можливого впливу на структуру білка певної хвороби.

Також алгоритми машинного навчання використовуються при диференціюванні генних стадій, визначаючи обставини, які змушують гени мутувати від нормального до хворобливого стану.

Поширений напрям використання машинного навчання для запобігання захворювання, коли прогнозне моделювання актуальне для ранньої діагностики захворювань та їх профілактики [4].

Таким чином, алгоритми машинного навчання дозволяють вирішити коло важливих питань. Але разом із тим можна говорити про існуючі проблеми, які пов'язані з природою даних біоінженерії. Відомо, що дані, які використовуються в машинному навчанні, представлені наступними типами: числові (безперервні, дискретні), категоріальні (порядкові, номінальні). Як правило, такі дані характеризуються великим розміром набору даних, наявністю категоріальних даних та слабкою репрезентативністю, складні в інтерпретації і обробці і призводять до складності у побудові моделей машинного навчання.

При використанні інформаційних технологій машинного навчання велику роль відграє інтелектуальний аналіз даних та розробка алгоритмів машинного навчання для подальшого їх використання.

Взагалі, в біомедицині наразі представлене програмне забезпечення з відкритим доступом до використання алгоритмів машинного навчання для передбачення параметрів молекул протеїнів, в основному пов’язаних зі структурними особливостями. На рисунку 1.1 наведений результат роботи системи AlphaFold, з відображеною передбаченою 3D-структурою білків.

AlphaFold враховує контакт між частинами амінокислотами. AlphaFold використовує згорткову нейронну мережу для прогнозування відстані між парами амінокислот вхідної послідовності. Ці відстані використовуються для формулювання функції протеїну. Мінімізація цієї функції за допомогою градієнтного спуску призводить до розуміння можливої структури білка, яка має найнижчу енергію та відповідає реальним білковим структурам.

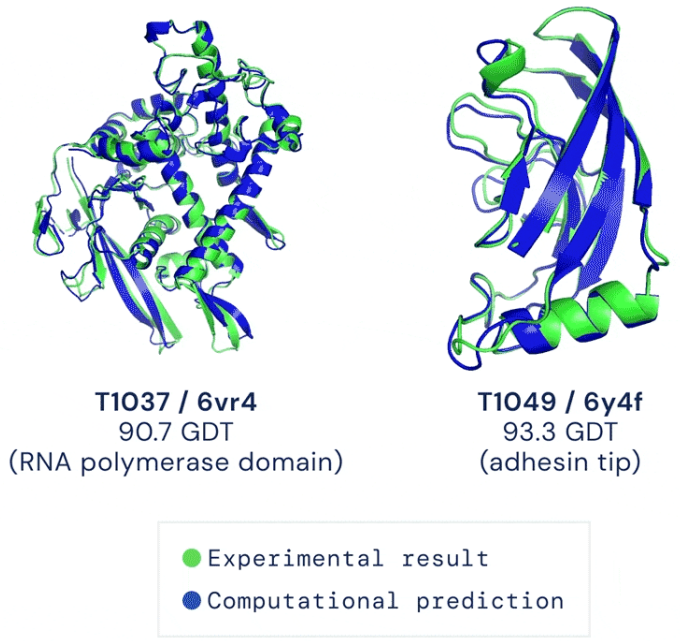

Рисунок 1.1 – Результат роботи програмного забезпечення AlphaFold з використанням машинного навчання для передбачення структури білків

Програмне забезпечення AlphaFold доступне на веб-сайті (див. рисунок.1.2) https://www.deepmind.com/.

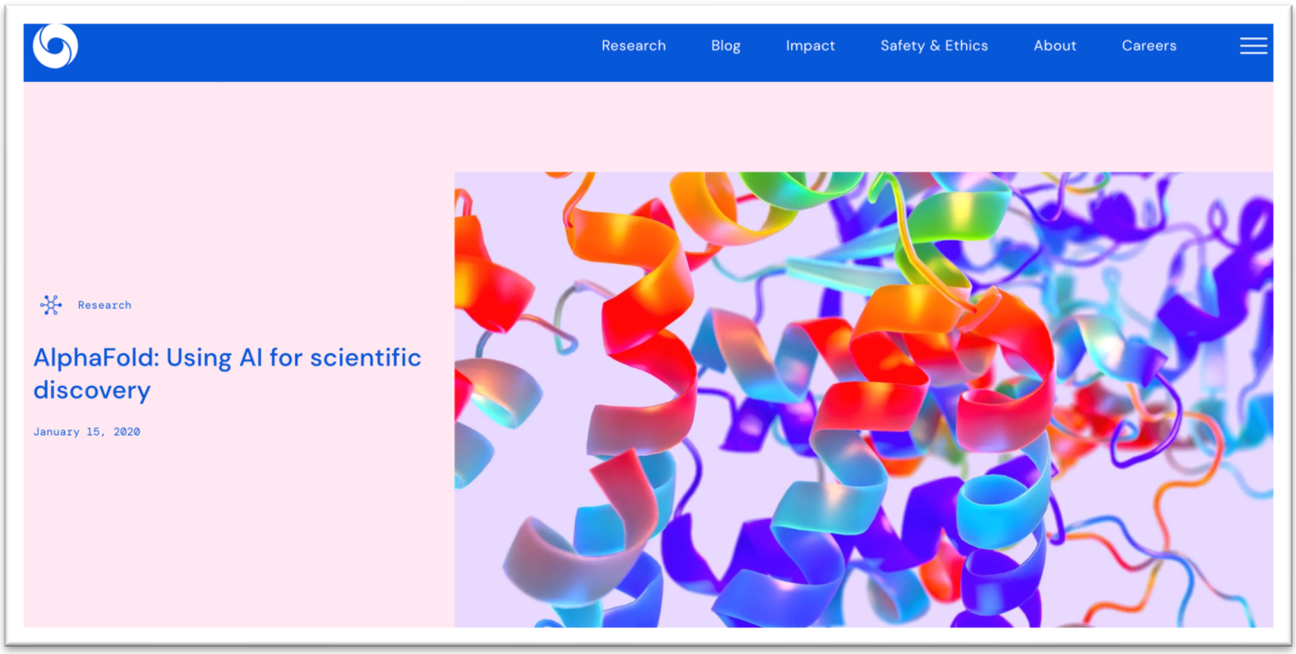

Рисунок 1.2 – Веб-сайт для наукових досліджень з використанням алгоритмів машинного навчання AlphaFold

Для візуалізації молекул протеїнів використовують PyMOL https://pymol.org/ (рис.1.3). Даний програмне середовище дозволяє створювати візуальне зображення по отриманим вхідним даним, що описують структуру протеїна. Дана система візуалізації молекул дозволяє створювати високоякісні тривимірні зображення як малих молекул, так і біологічних макромолекул білків.

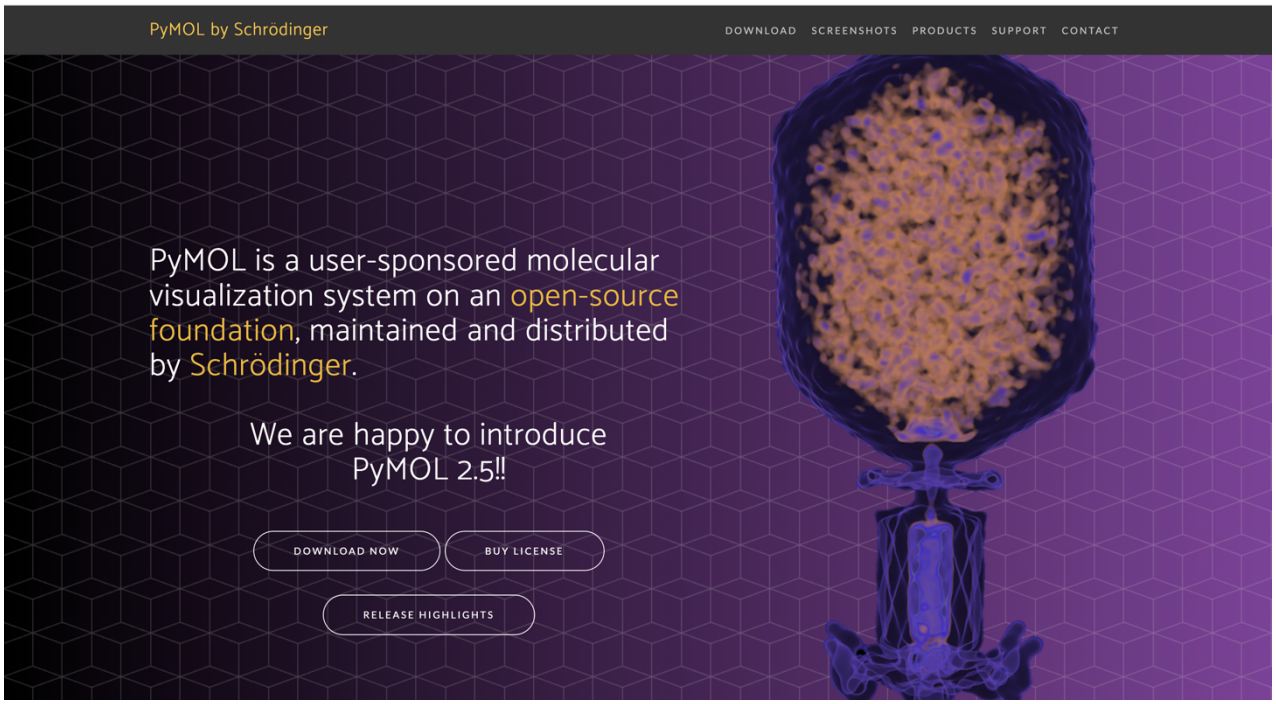

Рисунок 1.3 – Програмне забезпечення для візуалізації молекул білків

1.2. Огляд алгоритмів машинного навчання

Машинне навчання

На даний момент існує багато різних типів алгоритмів машинного навчання.

Алгоритми машинного навчання поділяються на типи:

- навчання з учителем,
- навчання без учителя (задачі кластерування),

- напівконтрольоване навчання,
- навчання з підкріпленням - навчання на основі попередніх результатів для досягнення максимальної користі.

Задачі навчання з учителем [8] представлені задачами регресії та класифікації. Для задач регресії найчастіше використовуються класичні алгоритми машинного навчання:

- Лінійна регресія
- Логістична регресія
- Дерево рішень
- Алгоритм SVM
- Наївний алгоритм Байєса
- Алгоритм KNN
- Алгоритм випадкового лісу
- Алгоритми зменшення розмірності

Для підвищення якості моделей використовуються також їх ансамблеві комбінації (бегінг, бустинг).

Глибоке навчання

Глибоке навчання [9] — це техніка машинного навчання, яка є важливим елементом науки про дані і включає статистику та прогнозне моделювання. За допомогою глибокого навчання полегшується процес збирання, аналізу та інтерпретування великих обсягів даних. Глибоке навчання можна розглядати як спосіб автоматизації прогнозної аналітики. У той час як традиційні алгоритми машинного навчання є лінійними, алгоритми глибокого навчання зібрані в ієрархію, де зростає складність і абстракція (рис.1.4).

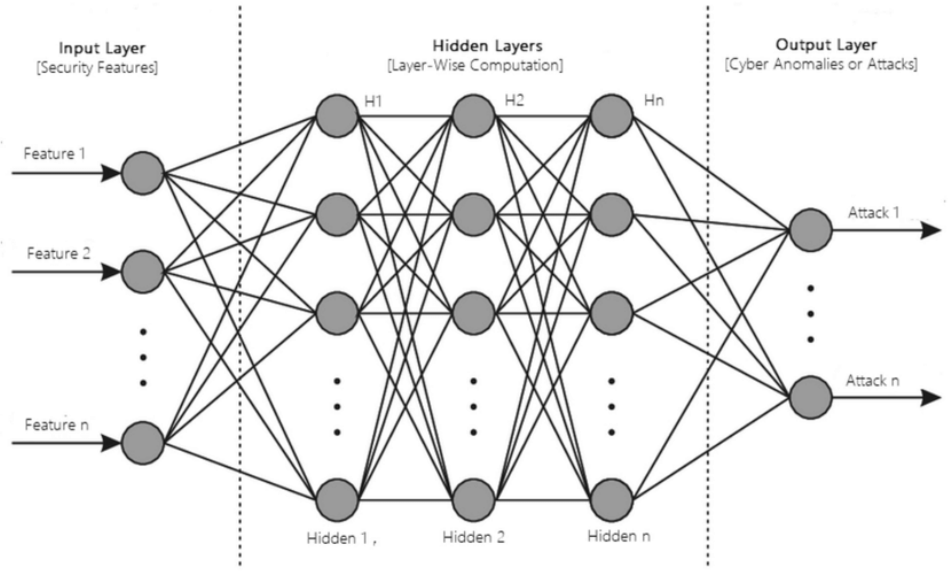

Рисунок 1.4 – Архітектура алгоритму глибокого навчання

Нейронні мережі

Нейронні мережі, також відомі як штучні нейронні мережі або імітовані нейронні мережі, є підмножиною машинного навчання та основою алгоритмів глибокого навчання. Їх назва та структура мають аналогію з людським мозком, імітуючи спосіб, яким біологічні нейрони передають сигнали один одному.

Штучні нейронні мережі складаються з вузлових шарів, що містять вхідний рівень, один або більше прихованих шарів і вихідний рівень. Кожен вузол, або штучний нейрон, з'єднується з іншим і має відповідну 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 [11].

Візуальна діаграма вхідного рівня, прихованого шару і вихідного рівня нейронної мережі прямого зв'язку наведена на рисунку 1.5.

Кожен окремий вузол можна представити як власну модель лінійної регресії, що складається з вхідних даних, вагових коефіцієнтів, зміщення (або порогу) і вихідних даних.

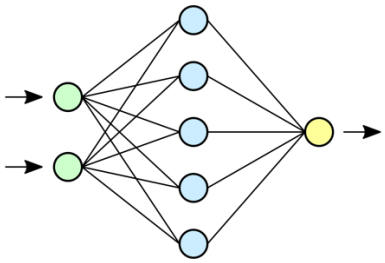

Рисунок 1.5 – Архітектура нейронної мережі

Нейронні мережі можна поділити на різні типи залежно від задач. Найпоширеніші типи нейронних мереж, які зустрісаються для типових випадків використання наведені нижче.

Персептрон — найстаріша нейронна мережа, створена Френком Розенблатом у 1958 році. Вона складається з одного нейрона і є найпростішою формою нейронної мережі (рис.1.6) [13].

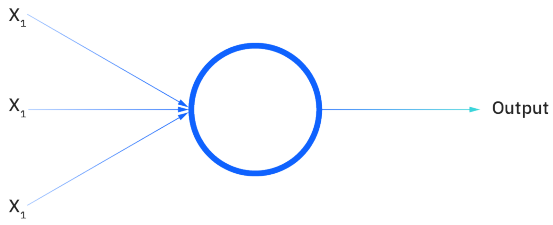

Рисунок 1.6 – Модель нейронної мережі Персептрона

Наступний тип нейронних мереж - нейронні мережі прямого зв’язку (повнозв’язані), або багатошарові персептрони, складаються з вхідного шару, прихованого шару або шарів і вихідного шару. Хоча ці нейронні мережі також зазвичай називають багатошаровим персептроном MLP, важливо зазначити, що насправді вони складаються з сигмовидних нейронів. Дані зазвичай подаються в ці моделі для їх навчання, вони використовуються для комп’ютерного зору, обробки мови та інших нейронних мереж.

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

образів. Ці мережі використовують принципи лінійної алгебри, зокрема множення матриць, щоб ідентифікувати шаблони в зображенні (рис.1.7).

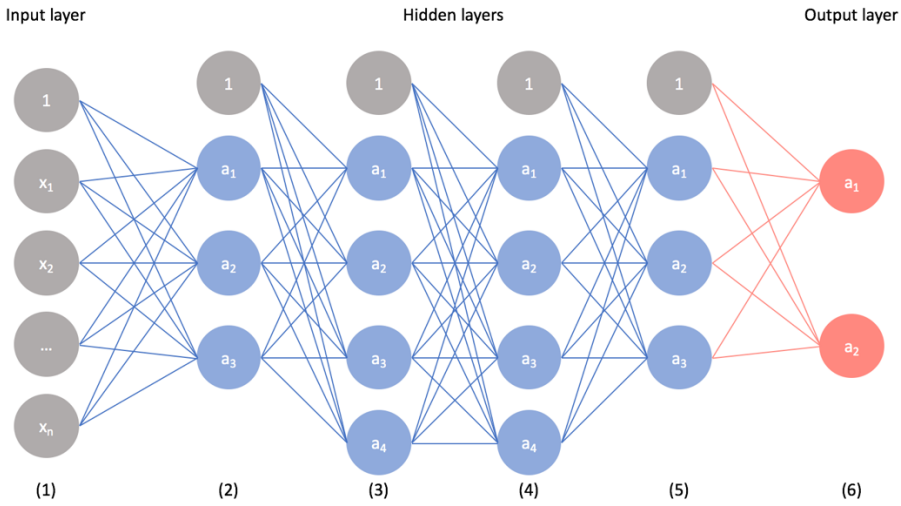

Рисунок 1.7 – Схема згорткової нейронної мережі

Рекурентні (згорткові) 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.

Також відомі багато інших типів нейронних мереж: мережі короткострокової пам'яті LSTM, генеративні GAN, двоспрямовані мережі та інші.

1.3. Постановка завдання

В результаті аналітичного аналізу літературних джерел можна зробити висновок про доцільність і ефективність використання алгоритмів машинного навчання в біоінженерії.

Метою роботи була розробка інформаційної технології машинного навчання на прикладі біомедичних даних з метою прогнозування їх параметрів (функції протеїну, епістазу). В якості вхідних даних необхідно використати

експериментальні дані для різних білків з одиночними та подвійними мутаціями.

Інформаційна технологія повинна включати:

- підготовку, аналіз та обробку вхідних даних, їх доповнення з метою формування вхідного математичного опису,
- підбір параметрів навчальної матриці,
- застосування алгоритмів машинного навчання для створення моделей машинного навчання, розробку та програмну реалізацію їх ансамблів, а також використання нейромереж для прогнозування параметра фітнес-функції або епістаза протеїнів на навчальній матриці,
- тестування моделей машинного навчання на тестовій вибірці матриці,
- аналіз отриманих результатів.

Програмну реалізацію здійснити з використанням мови програмування Python та бібліотек, пов’язаних з обробкою та візуалізацією даних, використанням моделей машинного навчання.

2 ВИБІР МЕТОДІВ РЕАЛІЗАЦІЇ ПОСТАВЛЕНОЇ ЗАДАЧІ

2.1 Вибір програмних засобів

Мова програмування Python

Можна стверджувати, що не існує єдиної мови програмування, яка б дозволяла вирішити всі задачі машинного навчання. Проте, безумовно, є деякі мови програмування, які більш підходять для таких завдань. Вважається, що найкраще використовувати Python, але також для завдань статистичного аналізу даних дуже популярна мова R [14].

Python — це мова програмування, яка підтримує створення широкого спектру програм. Розробники вважають її накращим вибором для проектів штучного інтелекту, машинного та глибокого навчання.

Мова має величезну кількість бібліотек і фреймворків, які полегшують кодування та значно економить час.

Найпопулярнішими бібліотеками є NumPy, яка використовується для наукових розрахунків; SciPy для складніших обчислень; і scikit для вивчення інтелектуального аналізу даних. Ці бібліотеки працюють разом із потужними фреймворками, наприклад, TensorFlow, що необхідно для проектів машинного та глибокого навчання.

Код Python є лаконічним і читабельним. Завдяки простому синтаксису розробка додатків на Python є швидкою порівняно з багатьма мовами програмування. Крім того, це дозволяє розробнику тестувати алгоритми без їх впровадження [15].

Python надає багато інструментів для перевірки коду та тестування. Розробники можуть швидко перевірити правильність і якість коду. Проекти штучного інтелекту, як правило, займають багато часу, тому потрібне добре структуроване середовище для тестування та перевірки на помилки.

Python постачається з великою різноманітністю бібліотек. Деякі з цих фреймворків пропонують хороші інструменти візуалізації. У штучному

інтелекті, машинному та глибокому навчанні важливо подавати дані в зручному для читання форматі. Тому Python є ідеальним вибором для реалізації цієї функції.

Деякі бібліотеки, такі як Matplotlib, дозволяють дослідникам даних створювати діаграми, гістограми та графіки для кращого представлення даних і візуалізації. На рисунку 2.1 наведені популярні бібліотеки візуалізаії даних.

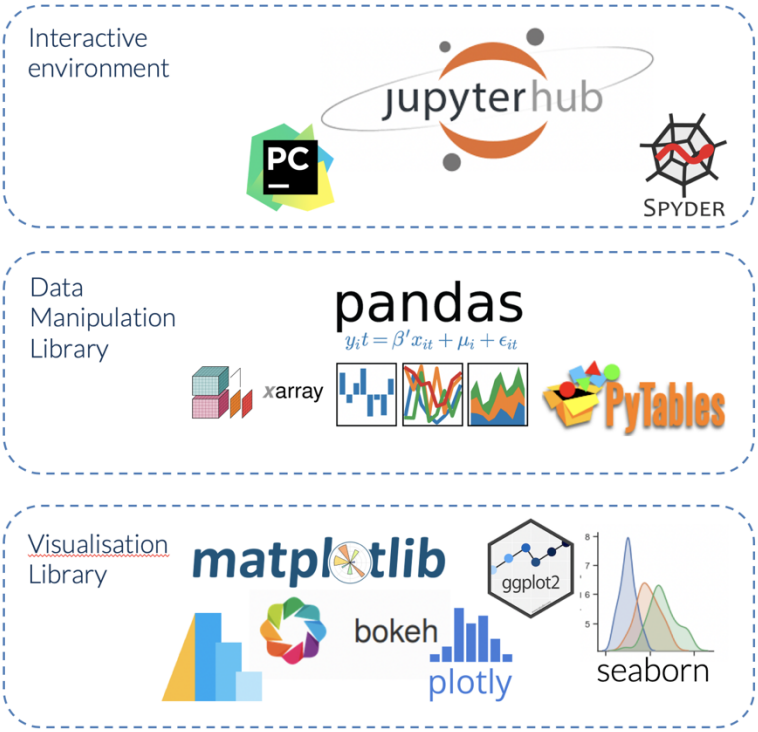

Рисунок 2.1 – Стек бібліотек для візуалізації даних Python

Бібліотеки і фреймворки Python

Мова програмування Python представлена багатьма важливими бібліотеками і фремворками [16]. Для виконання поставленого завдання буди обрані наступні з них:

- TensorFlow,
- Scikit-Learn,
- Keras,
- Numpy,

- SciPy,
- Pandas,
- Re,
- Seaborn,
- Matplotlib,
- Plotly та інші.

Scikit-learn - один із найбільш широко використовуваних пакетів Python для науки про дані і машинного навчання. Він дозволяє виконувати безліч операцій і надає доступ до великої кількості алгоритмів. Scikit-learn також пропонує документацію про свої класи, методи та функції, а також опис використовуваних алгоритмів.

Scikit-Learn підтримує попередню обробку даних, зменшення розмірності, вибір моделі, регресії, класифікації, кластерний аналіз.

Scikit-learn не реалізує все, що пов'язано з машинним навчанням. Наприклад, він не має комплексної підтримки для нейронних мереж та навчання с підкріпленням (reinforcement learning).

Для побудови алгоритмів машинного навчання були використані TensorFlow, Scikit-Learn та Keras. Бібліотека TensorFlow була розроблена Google у співпраці з Brain Team. TensorFlow є частиною майже кожної програми Google для машинного навчання. TensorFlow працює як обчислювальна бібліотека для написання нових алгоритмів, які включають велику кількість тензорних операцій, оскільки нейронні мережі можна легко виразити у вигляді обчислювальних графів, їх можна реалізувати за допомогою TensorFlow як серії операцій над тензорами. Крім того, тензори — це N-вимірні матриці, які представляють дані. TensorFlow оптимізовано для швидкості, він використовує методи для швидких операцій лінійної алгебри.

Keras — це бібліотека Python, що включає API для роботи з нейронними мережами та фреймворками глибокого навчання [14]. Keras містить методи та компоненти на основі Python для роботи з різними програмами глибокого

навчання. Keras — це проста у використанні, але потужна бібліотека глибокого навчання для Python. Бібліотека націлена на оперативну роботу з нейромережами глибинного навчання, при цьому спроектована так, щоб бути компактною, модульною та розширюваною. Бібліотека Keras містить численні реалізації широко застосовуваних блоків нейронних мереж, таких як шари, цілі та передаточні функції, оптимізатори та безліч інструментів для покращення роботи із зображеннями та текстом.

Середовище розробки

Для роботи використовувалось інтегроване середовище розробки для наукового програмування мовами Python і R Anaconda (рис.2.2). Anaconda має відкритий вихідний код і є найпростішим засобом для розробки програмного забезпечення для обробки наукових даних та машинного навчання.

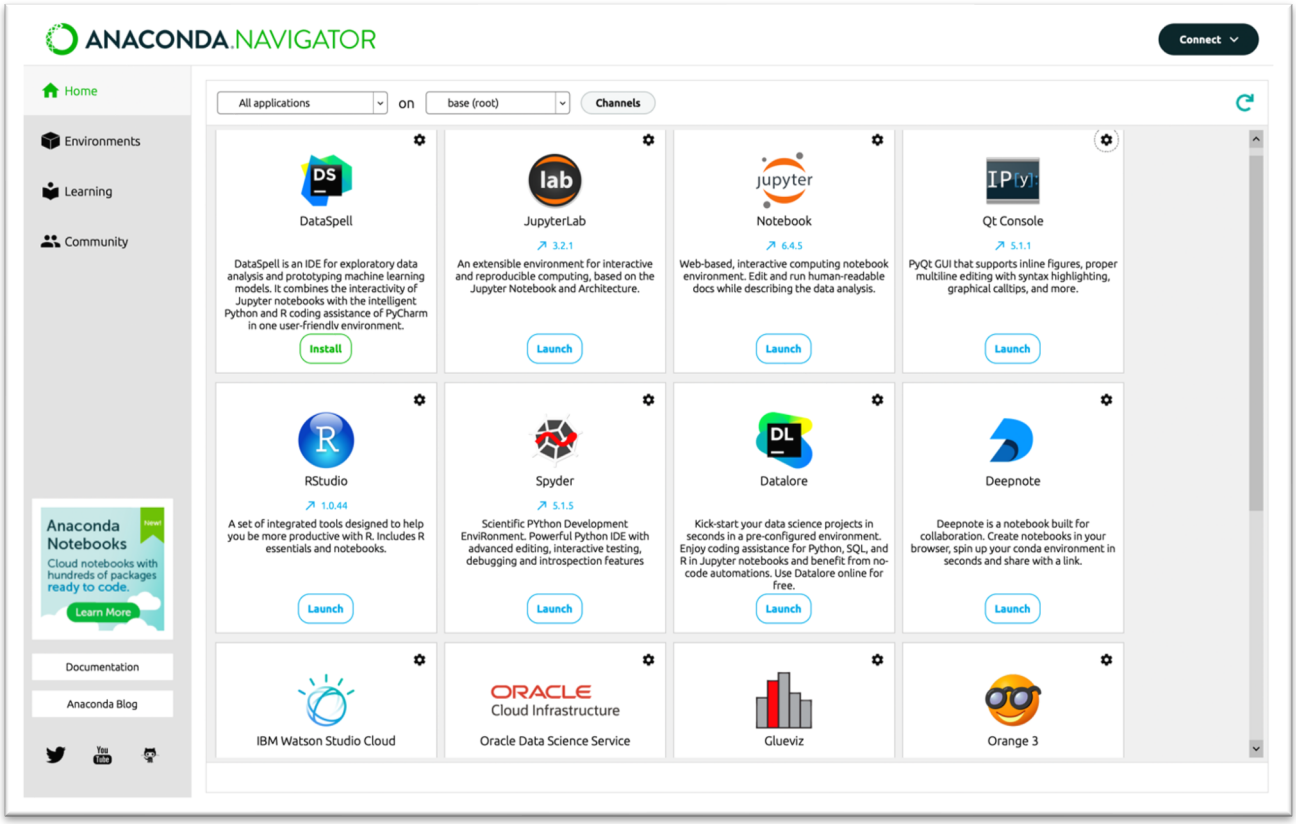

Рисунок 2.2 – Вікно середовища розробки Anaconda

Jupyter Notebook – веб-додаток з відкритим вихідним кодом, в якому можна відразу побачити результат виконання коду, Jupyter Notebook входить в пакет Anaconda.

Робота в Jupyter Notebook дозволяє писати код, додавати додаткові елементи в якості пояснень, виконувати блоки коду окремо

Використовувалось також ще одне середовище Spyder, Scientific Python Development Environment, наукове програмне забезпечення із зручним інтерфейсом, яке представлене у пакеті Anaconda.

2.2 Алгоритми розв’язання задачі регресії

Побудова моделей

Методи машинного навчання в основному поділяються на такі категорії:

• контрольоване навчання, алгоритми прогнозують значення невідомих змінних за допомогою наявних даних. Спочатку аналізується відомий навчальний набір даних, а потім використовується функція, яка робить прогнози щодо вихідних значень подібного набору даних [15].

• неконтрольоване навчання. Алгоритми навчаються лише з деякими вхідними зразками або мітками, тоді як вихід невідомий. Навчальна інформація не позначена. Отже, результат не завжди може бути правильним порівняно з навчанням під контролем.

• навчання з підкріпленням є технікою машинного навчання на основі зворотного зв'язку. Алгоритми досліджують дані, виконують дії та на основі своїх дій отримують винагороду як зворотний зв’язок. Метою агента навчання з підкріпленням є максимізація позитивних винагород.

• напівконтрольоване навчання - проміжна техніка як контрольованого, так і неконтрольованого навчання. Він виконує дії з наборами даних, які мають кілька міток, а також з даними без міток. Однак зазвичай він містить немарковані дані. Це підвищує точність і продуктивність моделі машинного навчання.

В даній роботі поставлена задача використання алгоритмів машинного контрольованого навчання для задачі регресії. Для задач регресії було обрано наступні алгоритми машинного навчання.

Дерева рішень (Decision Trees) — це непараметричний контрольований метод навчання, який використовується для задач класифікації та регресії (рисю2.3). Ціль складається в тому, щоб створити модель, яка передбачає значення цільової змінної, вивчаючи прості правила прийняття рішень, виведені з характеристик даних. Дерево можна розглядати як дискретно-постійне приближення. Концептуально дерева рішень - дуже простий алгоритм, достатньо однієї схеми, щоб зрозуміти принцип його роботи.

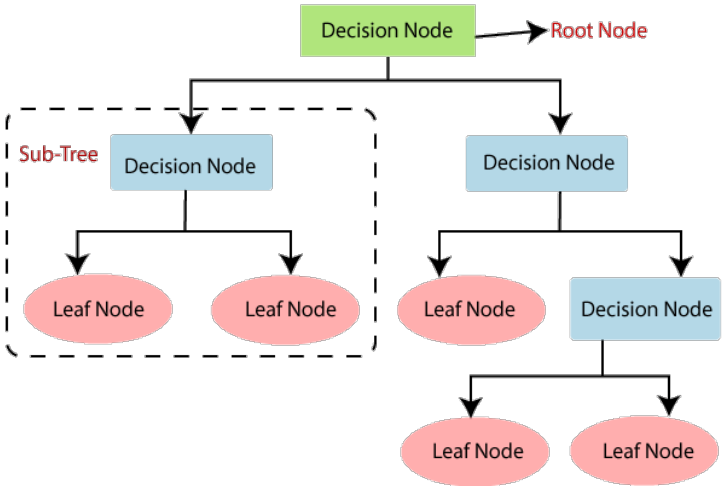

Рисунок 2.3 – Схема алгоритму дерев рішень

Випадковий ліс (Random Forest) — це метод ансамблевого навчання, заснований на деревах рішень. Створення випадкових лісів включає в себе кілька дерев рішень із використанням самоналаштовуваних наборів вихідних даних і випадкового вибору підмножини змінних на етапі дерева рішень. Потім модель збирає пакет прогнозів кожного дерева рішень. Модель враховує кількість прогнозів «за» і «проти» і виносить рішення, виходячи з більшості голосів (рис.2.4).

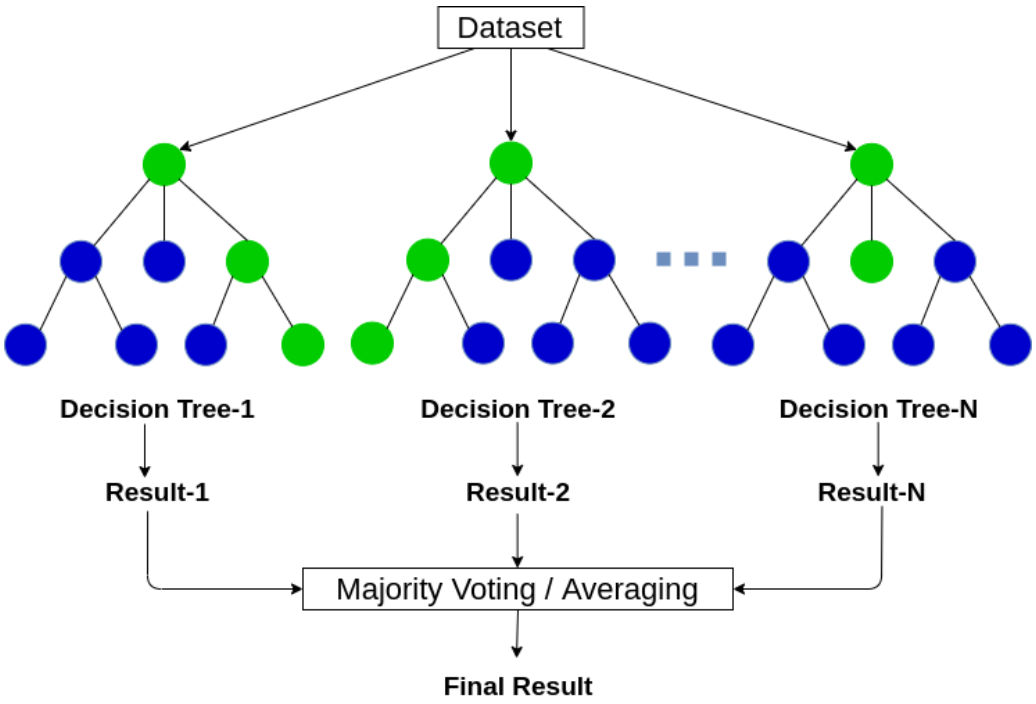

Рисунок 2.4 – Схема алгоритму випадкового лісу

Коли потрібно вирішити складну розрахункову задачу, використовують ансамблі. Ансамблі – поєднання відразу декількох алгоритмів, які вчаться одночасно і виправляють помилки іншого друга. На сьогоднішній день саме вони дають найбільш точні результати, тому саме їх частіше всього використовують усі великі компанії, для яких важлива швидка обробка великої кількості даних. Найкращий результат отримується, коли алгоритми в ансамблях максимально різні. Наприклад, регресія (Regression) і дерева рішень (Decision Trees) поєднуються відмінно.

Ансамблі моделей машинного навчання представлені так званим беггінгом та бустінгом. На рисунку 2.5 подана схематична відмінність дії таких алгоритмів.

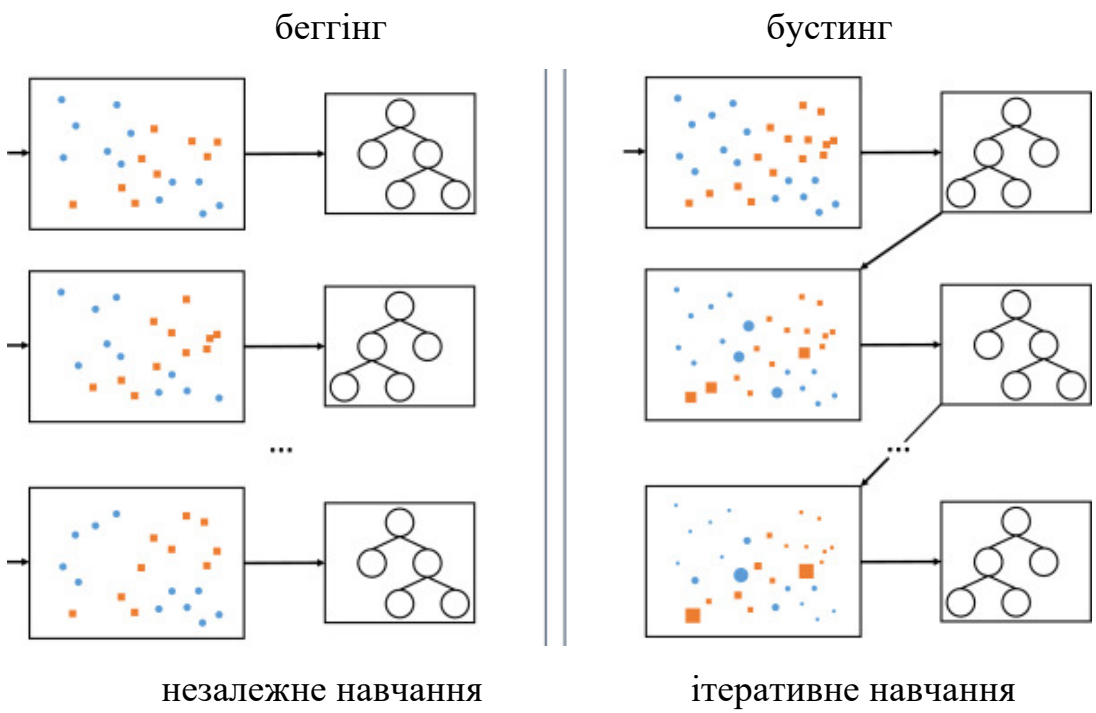

Рисунок 2.5 – Схема ансамблевих алгоритмів беггінгу та бустингу

AdaBoost, або адаптивний перехід, представляє собою ансамблевий алгоритм, який використовує методи збору прогнозів і підрахунку коефіцієнтів. AdaBoost схожий на випадкові ліси в тому сенсі, що прогнози беруться з багатьох дерев рішень, однак є відмінності. В AdaBoost використовуються дерева з однією вершиною і двома вузлами. Не всі створювані дерева мають право голосу для остаточного прогнозу. Ті, які часто помилялися, будуть менше впливати на остаточне рішення. Порядок розташування дерев важливий. Прогнози дерева направлені на зменшення помилок, зроблених попередніми.

Ансамблевий алгоритм Gradient Boost (градієнтний перебір) має свої особливості. Gradient Boost розглядає проблему оптимізації, де він використовує функцію втрат і намагається мінімізувати помилку за допомогою градієнтного спуску.

Дерева використовуються для прогнозування залишків (розділ між прогнозованими та реальними даними, тобто помилки). Алгоритм Gradient

Boost починається з побудови одного дерева, а наступні дерева націлені на зменшення залишків.

Глибоке навчання набуло величезної популярності в наукових обчисленнях, а його алгоритми широко використовуються галузями, які вирішують складні проблеми. Усі алгоритми глибокого навчання використовують різні типи нейронних мереж для виконання певних завдань.

Нейронна мережа складається зі штучних нейронів, також відомих як вузли. Ці вузли розташовані поруч один з одним у декілька шарів:

- вхідний шар,
- прихований шар або декілька шарів,
- вихідний шар.

Дані надають кожному вузлу інформацію у формі вхідних даних. Вузол множить вхідні дані на випадкові ваги, обчислює їх і додає зміщення. Нарешті, нелінійні функції, також відомі як функції активації, застосовуються для визначення того, який нейрон запускати (рис.2.6).

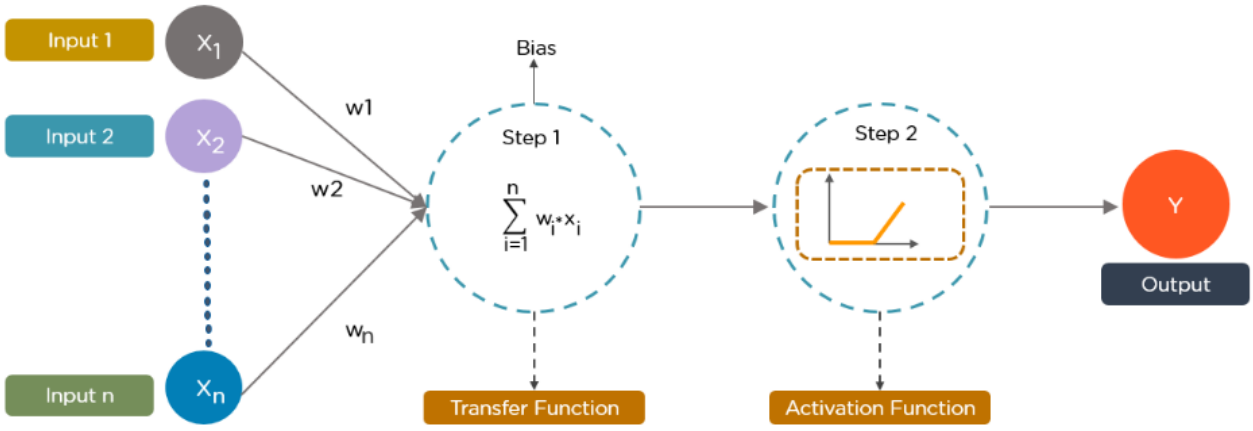

Рисунок 2.6 – Схема нейронної мережі з архітектурою глибокого навчання

Під час процесу навчання алгоритми використовують невідомі елементи у розподілі вхідних даних, щоб витягувати функції, групувати об’єкти та

виявляти корисні шаблони даних. Це відбувається на кількох рівнях із використанням алгоритмів для створення моделей.

Моделі глибокого навчання використовують кілька алгоритмів. Хоча жодна мережа не вважається ідеальною, деякі алгоритми краще підходять для виконання конкретних завдань. Щоб вибрати правильні алгоритми, добре отримати чітке розуміння всіх основних алгоритмів.

Найпопулярніші алгоритми глибокого навчання наступні:

- згорткові нейронні мережі (CNN)
- мережі довготривалої короткострокової пам'яті (LSTM)
- повторювані нейронні мережі (RNN)
- генеративні змагальні мережі (GAN)
- радиально-базисні функціональні мережі (RBFN)
- багатошарові персептрони (MLP)
- самоорганізуючі карти (SOM)
- мережі глибокої віри (DBN)
- обмежені машини Больцмана (RBM)
- автокодери.

До алгоритмів глибокого навчання, які були застосовані, відносяться наступні:

- повнозв’язані нейронні мережі.

Характеризується тим, що кожен нейрон попереднього шару пов'язаний з кожним нейроном наступного шару. Сигнал поширюється від вхідного шару до вихідного, не утворюючи зворотних зв'язків. Математична модель нейронної мережі прямого зв’язку наведена на рисунку 2.7.

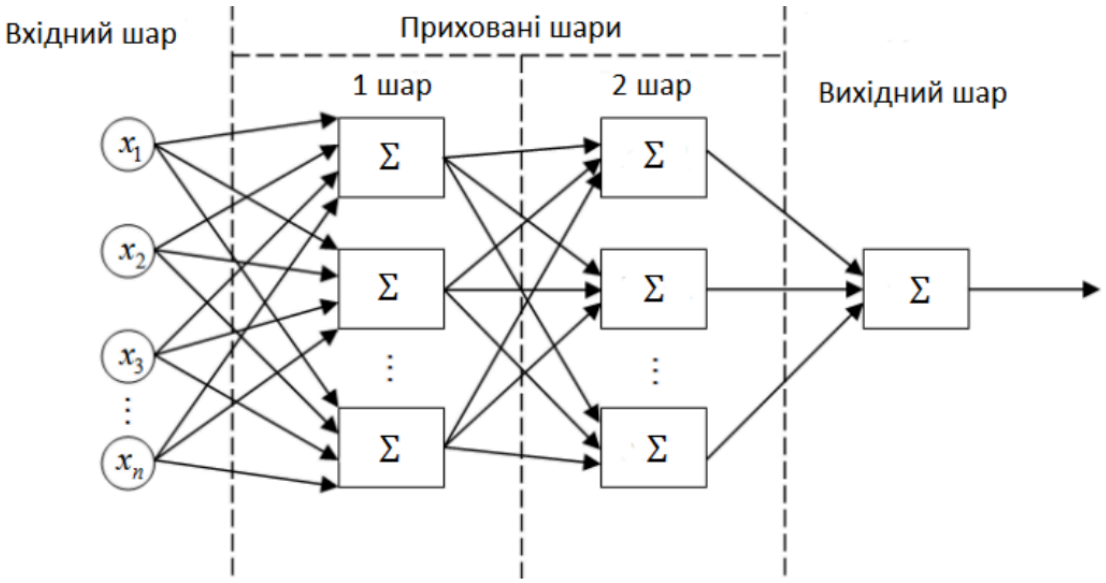

Рисунок 2.7 – Математична модель нейронної мережі прямого зв’язку

- згорткові нейронні мережі.

Згорткові нейромережі відомі як ConvNets, складаються з кількох шарів і в основному використовуються для обробки зображень. Янн ЛеКун розробив у 1988 році, на той час такі мережі називалися LeNet і використовувались для розпізнавання таких символів, як поштові індекси та цифри.

Згорткова нейронна мережа складається з кількох блоків, таких як шари згортки, шари об’єднання та повнозв’язані шари, і розроблена для автоматичного й адаптивного вивчення просторових ієрархій функцій за допомогою алгоритму зворотного поширення.

CNN мають кілька рівнів, які обробляють і витягують функції з даних. Шар згортки, який має кілька фільтрів для виконання операції згортки. Випрямлений лінійний блок (ReLU) використовується для виконання операцій над елементами. Далі виправлена матриця ознак надходить у шар об’єднання (pooling layer). Об’єднання — це операція зменшення вибірки, яка зменшує розміри матриці ознак. Потім шар об’єднання перетворює отримані двовимірні масиви з об’єднаної матриці ознак в єдиний довгий безперервний лінійний

вектор шляхом його зведення. Повністю пов’язаний шар утворюється, коли сплощена матриця з шару об’єднання подається як вхідний сигнал.

- рекурентні нейронні мережі.

Архітектура традиційної рекурентної нейронні мережі є класом нейронних мереж, які дозволяють використовувати попередні вихідні дані як вхідні, маючи приховані стани.

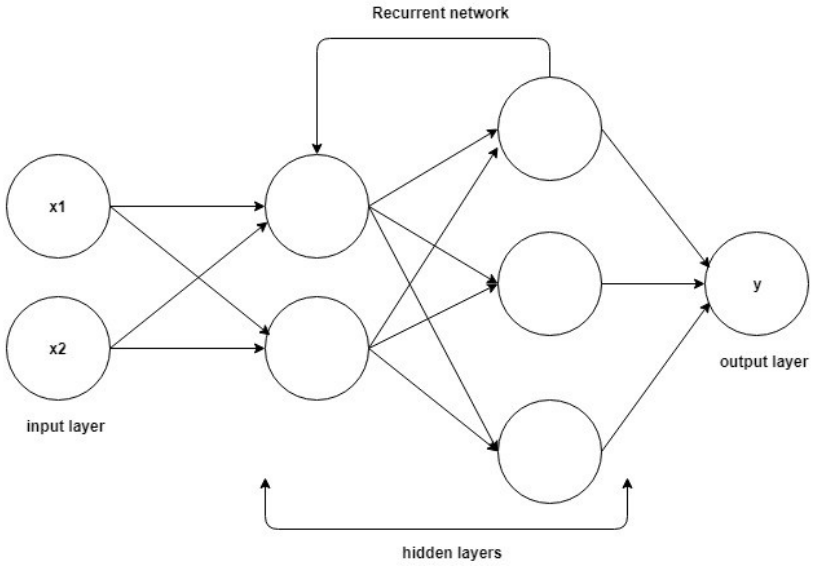

Рисунок 2.8 –Модель рекурентної нейронної мережі

Рекурентні нейронні мережі (RNN) — це клас нейронних мереж, які підходять для моделювання послідовних даних, RNN зазвичай використовуються для підписів до зображень, аналізу часових рядів, обробки природної мови, розпізнавання рукописного тексту та машинного перекладу (рис.2.8).

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1. Опис вхідних даних. Розробка інформаційної технології машинного навчання

Машинне навчання збільшує здатність вивчати та прогнозувати поведінку залежності між параметрами протеїнових DNA-послідовностей. Прогноз фітнес-функції або значення епістаза дозволяє розробити нові протеїнові послідовності з особливою структурою та корисними властивостями.

Традиційно аналіз даних є найважливішою і складною частиною. Основна проблема забезпечення повного аналізу даних для машинного навчання пов’язана з наявністю категоріальних та порядкових вхідних даних та відносно малим розміром наборів даних.

Як було зазначено, машинне навчання є ефективним методом дослідження параметрів протеїнів. Процес побудови інформаційної технології машинного навчання складається з наступних кроків (рисунок 3.1).

Рисунок 3.1 – Модель інформаційної технології

Підготовка даних є одним із найскладніших кроків у проектах машинного навчання. Підготовка даних — це процес перетворення вихідних даних. Це дозволяє розуміти вхідні дані, працювати з функціями, очищати та покращувати якість даних. Необроблені дані не можна використовувати безпосередньо. Крім того, алгоритми машинного навчання потребують лише числових даних.

Підготовка даних включає:

- очищення даних. Виявлення та виправлення помилок або помилок у даних (видалення відсутніх даних, викидів, повторюваних значень і помилок експерименту),
- вибір функцій цілі: визначення найбільш релевантних для завдання змінних,
- візуалізація даних, щоб зрозуміти, як вони структуровані, зв'язок між різними змінними,
- підбір параметрів, запобігання мультиколінеарності.
- перетворення даних: зміна масштабу або розподілу змінних (наприклад, нормалізація, стандартизація), кодування,
- отримання нових змінних із доступних даних (додавання відомих даних таблиці, розширення набору даних),
- зменшення розмірності: створення компактних проекцій даних.

Розбиття набору даних на два набори: набір для навчання та набір для тестування. Навчальний набір – це набір вхідних даних, на якому модель навчається (80%). Тестовий набір використовується для перевірки точності моделі після навчання і складає 20%.

Вхідні дані представлені на рисунку 3.2.

	delta_bulkiness	delta_polarity	delta_hydrophobicity	mean_aa1	mean_aa2	mutated_aa1	mutated_aa2	pos1	pos2	distance	epistasis
0	7.94	0.3	2.0	0.394965	0.466941	C	I	20	71	10.968905	-0.448487
1	8.28	4.0	3.1	0.495651	0.362290	D	G	20	126	18.974899	-0.368747
2	2.21	3.8	2.7	0.495651	0.433522	D	S	20	80	22.593917	-0.493776
3	2.60	2.5	1.0	0.495651	0.446184	D	R	20	23	5.323774	-0.563905
4	4.10	3.1	2.7	0.356815	0.113844	E	S	20	85	19.795908	0.171792
...	...	...	...	...	...	...	...	...	...	...	...
5373	6.34	0.3	0.3	0.377707	0.441704	C	F	104	141	11.300248	-0.447156
5374	18.17	3.1	4.6	0.182654	0.567471	G	V	104	144	15.954608	-0.563484
5375	18.00	3.8	4.9	0.182654	0.491280	G	I	104	140	12.871063	-0.344918
5376	0.00	0.3	0.7	0.447980	0.325367	L	I	104	148	19.589989	-0.243530
5377	8.85	6.2	2.6	0.521134	0.408916	W	N	104	113	14.395373	-0.428438

5378 rows × 11 columns

Рисунок 3.2 – Приклад вхідних даних

Етап розробки моделей машинного навчання включає проектування алгоритмів машинного навчання та налаштування їх гіперпараметрів. Гіперпараметри — це параметри, що дозволяють керувати процесом навчання моделі. Наприклад, в нейромережах можна задати кількість закритих шарів і кількість вузлів у кожному шарі. Продуктивність моделі в великій мірі залежить від гіперпараметрів.

Моделі машинного навчання можуть мати різні гіперпараметри, пошук найкращої їх комбінації можна розглядати як окрему проблему. Налаштування гіперпараметрів допомагає знайти найкращу їх сукупність:

• вибір алгоритму оптимізації (наприклад, оптимізатор Адама),

• вибір функції активації на рівні нейронної мережі (наприклад, ReLU),

• вибір функції втрат, яку використовуватиме модель (MSE),

• кількість прихованих шарів у нейронній мережі,

• кількість одиниць активації в кожному шарі,

• кількість ітерацій (епох) у навчанні нейронної мережі,

• розмір ядра або фільтра в згорткових шарах, тощо.

Навчання моделей здійснюється на навчальній матриці. Під час навчання підготовлені дані передаються моделям машинного навчання з метою

знаходження закономірностей та можливістю передбачення даних на невідомих даних. В процесі навчання можливий підбір гіперпараметрів і переналаштування архітектури нейронної мережі. Оптимізація гіперпараметрів — це процес пошуку конфігурацій гіперпараметрів, що приводять до кращої продуктивності.

Оцінювання моделей машинного навчання здійснюється шляхом перевірки їх продуктивності на раніше невідомих даних тестової вибірки. Оцінка моделей машинного навчання на тренувальній матриці даних складається з вибору процедури оцінки моделі та показників ефективності для оцінки навичок прогнозування моделі.

У задачах машинного навчання для оцінки якості моделей і порівняння різних алгоритмів використовуються метрики. Були обрані стандартні показники оцінки - метрики для задачі регресії:

- середня абсолютна похибка (MAE),

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|a(x_i) - y_i|,$$

де y_i - вихідне значення

$a(x_i)$ - прогнозоване значення регресійної моделі

n - кількість даних

- середня квадратична помилка (MSE),

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}(a(x_i) - y_i)^2,$$

де y_i - вихідне значення

$a(x_i)$ - прогнозоване значення з регресійної моделі

n - кількість даних

- середня квадратична помилка (RMSE),

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(a(x_i) - y_i)^2}$$

- коефіцієнт детермінації:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(\mathrm{a(xi)} - \mathrm{yi})^2}{\sum_{i=1}^{n}(\mathrm{y}_i - \bar{\mathrm{y}}_\imath)^2}$$

- функція втрат (для нейронних мереж).

Для наочності було вирішено використовувати коефіцієнти кореляції Пірсона і Спірмана між початковими тестовими та прогнозованими значеннями. Для правильної оцінки прогнозної ефективності моделей проводилась перехресна перевірка cross-validation (рис.3.3). Також розроблений і використаний алгоритм автоматичний підбору найкращих гіперпараметрів. Деякі прогностичні моделі були поєднали в ансамблі.

Рисунок 3.3 – Схема методу кросс-валідації

Остаточну модель машинного навчання можна використовувати для прогнозування нових наборів даних. Зазвичай використовується тестовий набір даних матриці ознак для прогнозування числових значень функції цілі (функція протеїну, епістаз).

Заключним етапом проектування інформаційної технології машинного навчання є аналіз отриманих результатів і порівняння їх з відомими або очікуваними даними.

3.2. Програмна реалізація

Програмна реалізація здійснювалась у середовищі Jupiter Notebook з використанням мов Python та необхідних бібліотек. В якості вхідних даних були взяті різні набори даних для деяких варіантів білків. У випадку використання вхідних даних з одиночними мутаціями амінокслот функцією цілі було обрано функцію протеїну, у випадку використання наборів даних з подвійними мутаціями – епістаз.

Первісна обробка даних включала роботу з категоріальними даними (рис.3.3), а також пошук і видалення дублікатів, пустих і неінформативних даних.ї

```
dd=df.drop(['n_aa_substitutions_syn'],axis=1)
dd['aa_mutation1'] = dd['aa_mutation_syn'].apply(lambda x: x.split(' ')[0])
dd['aa_mutation2'] = dd['aa_mutation_syn'].apply(lambda x: x.split(' ')[1])
dd['wt_aa1'] = dd['aa_mutation1'].apply(lambda x: x[0:1])
dd['wt_aa2'] = dd['aa_mutation2'].apply(lambda x: x[0:1])
dd['mutated_aa1'] = dd['aa_mutation1'].apply(lambda x: x[-1:])
dd['mutated_aa2'] = dd['aa_mutation2'].apply(lambda x: x[-1:])
dd['pos1'] = dd['aa_mutation1'].apply(lambda x: x[1:-1])
dd['pos1'] = dd['pos1'].astype(int)
dd['pos1'] = dd['pos1']-136
dd['pos2'] = dd['aa_mutation2'].apply(lambda x: x[1:-1])
dd['pos2'] = dd['pos2'].astype(int)
dd['pos2'] = dd['pos2']-136
dd = dd.query('pos1 >0 and pos1<155')
dd = dd.query('pos2 >0 and pos1<155')
dd.reset_index(drop=True, inplace=True)
```

Рисунок 3.3 – Приклад коду обробки вхідних даних

	delta_bulkiness	delta_polarity	delta_hydrophobicity	mean_aa1	mean_aa2	distance	epistasis	mutated_aa1_A	mutated_aa1_C	mutated_aa1_D
0	7.94	0.3	2.0	0.394965	0.466941	10.968905	-0.448487	0	1	0
1	8.28	4.0	3.1	0.495651	0.362290	18.974899	-0.368747	0	0	1
2	2.21	3.8	2.7	0.495651	0.433522	22.593917	-0.493776	0	0	1
3	2.60	2.5	1.0	0.495651	0.446184	5.323774	-0.563905	0	0	1
4	4.10	3.1	2.7	0.356815	0.113844	19.795908	0.171792	0	0	0
...	...	...	...	...	...	...	...	...	...	...
5373	6.34	0.3	0.3	0.377707	0.441704	11.300248	-0.447156	0	1	0
5374	18.17	3.1	4.6	0.182654	0.567471	15.954608	-0.563484	0	0	0
5375	18.00	3.8	4.9	0.182654	0.491280	12.871063	-0.344918	0	0	0
5376	0.00	0.3	0.7	0.447980	0.325367	19.589989	-0.243530	0	0	0
5377	8.85	6.2	2.6	0.521134	0.408916	14.395373	-0.428438	0	0	0

5378 rows × 331 columns

Рисунок 3.4 – Приклад вхідних даних після перетворення категоріальних даних

Як було зазначено, дані характеризуються наявною кількістю категоріальних ознак. Для нечислових даних застосовувався метод перетворення в бінарний вид за допомогою методу one-hot encoding. Вибору інформативних ознак було приділено особливу увагу. Серед традиційних методів був використаний підхід визначення взаємної інформації і його реалізація за допомогою методу mutual_info_regression. Взаємна інформація між двома випадковими величинами є невід'ємною величиною, яка вимірює залежність між змінними. Вона дорівнює нулю тоді і тільки тоді, коли дві випадкові змінні незалежні, а більші значення означають більшу залежність. Тобто взаємна інформація визначає можливість побудувати вимірний зв'язок між ознакою та функцією цілі. Нижче наведений приклад використання функції mutual_info_regression:

```
def scalling(X_train,X_test):
    scaler = StandardScaler()
    scaler.fit(X_train)
    X_train_scaled = scaler.transform(X_train)
    X_test_scaled = scaler.transform(X_test)
    return X_train_scaled, X_test_scaled
# feature selection
def select_features(X_train, y_train, X_test):
        fs = SelectKBest(score_func=mutual_info_regression,k=10)
        fs.fit(X_train, y_train)
        X_train_fs = fs.transform(X_train)
        X_test_fs = fs.transform(X_test)
        return X_train_fs, X_test_fs, fs
```

Для швидкого підбору параметрів був використаний клас SelectKBest:

```
for k in num_features:
  # create pipeline
  model = RandomForestRegressor()
  fs = SelectKBest(score_func=mutual_info_regression, k=k)
  pipeline = Pipeline(steps=[('sel',fs), ('lr', model)])
  # evaluate the model
  cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)
  scores = cross_val_score(pipeline, X_train, y_train, scoring='neg_mean_absolute_error', cv=cv, n_jobs=-1)
  results.append(scores)
  # summarize the results
  print('>%d %.3f (%.3f)' % (k, mean(scores), std(scores)))
# plot model performance for comparison
pyplot.boxplot(results, labels=num_features, showmeans=True)
pyplot.show()
```

Вибір ознак – це метод, за допомогою якого можна вибрати ті характеристики в даних, які найбільше впливають на цільову змінну. Таким чином, можна обрати найкращі предиктори для цільової змінної. В свою чергу, використання алгоритмів підбору ознак зменшує перенавчання. Надлишкові дані впливають на можливість прийняття рішень. Також, як наслідок, підвищується точність та швидкість роботи алгоритму.

Набір ознак в залежності від задачі і вхідного набору даних представлений наступними характеристиками - функція протеїна, епістаз, позиція мутації, молекулярна маса, гідрофобність, полярність, заряд тощо.

Для візуалізації даних використовувались методи графічних бібліотек, були отримані дані про розподіл значень параметрів, також побудована теплова

карту ознак за допомогою метода heatmap(), графіки парної залежності pairplot() для підбору параметрів, придатних для машинного навчання.

При побудові нейромереж використовувався розроблений алгоритм автоматичного підбору гіперпараметрів на основі визначеної архітектури. Для запобігання перенавчанню, використовувався метод ранньої зупинки (клас EarlyStopping TensorFlow), який припиняє навчання, коли відстежуваний показник нейромережі перестає покращуватися.

```
model = k.Sequential()
model.add(k.layers.Dense(units=600, activation="relu"))
model.add(k.layers.Dense(units=124, activation="relu"))
model.add(k.layers.Dense(units=64, activation="relu"))
model.add(k.layers.Dense(units=1))
model.compile(loss="mse", optimizer="adam", metrics = 'mae')
model.save_weights("weigts.h5")
fit_res = model.fit(X_train_scaled,  y_train, epochs = 100,batch_size=16,  validation_split=0.2,
                    callbacks=EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=50))
mse, mae = model.evaluate(X_test_scaled, y_test, verbose = 0)
```

В результаті роботи алгоритму машинного навчання на прикладі повнозв'язаної нейромережі для передбачення значень епістазу молекул протеїнів з подвійними мутаціями дані представляються у вигляді визначених метрик та їх значень. На рисунку 3.5 якість навченої моделі із двома прихованими шарами визначається значеннями mae=0.06, loss (mse) = 0.00036.

```
Epoch 97/100
202/202 [==============================] - 1s 5ms/step - loss: 4.8936e-04 - mae: 0.0160 - val_loss: 0.0099 - val_mae:
0.0580
Epoch 98/100
202/202 [==============================] - 1s 5ms/step - loss: 3.9307e-04 - mae: 0.0145 - val_loss: 0.0099 - val_mae:
0.0563
Epoch 99/100
202/202 [==============================] - 1s 5ms/step - loss: 3.3298e-04 - mae: 0.0134 - val_loss: 0.0093 - val_mae:
0.0573
Epoch 100/100
202/202 [==============================] - 1s 5ms/step - loss: 3.6175e-04 - mae: 0.0139 - val_loss: 0.0101 - val_mae:
0.0569
mae= 0.059424277395009995
mse= 0.010952649638056755
43/43 [==============================] - 0s 3ms/step
127/127 [==============================] - 0s 2ms/step
```

Рисунок 3.5 – Результат роботи повнозв'язаної нейромережі

Таким чином, був розроблений і програмно реалізований стек моделей навчання з учителем для задачі регресії, використовуючи підходи різних авторів [4, 6, 7, 12,15]: Random Forest, DecisionTreeRegressor, Gradient Boosting Regressor, XGBRegressor, SVR. Також використовувались ансамблеві алгоритми Bagging і Boosting на їх основі.

В якості алгоритмів глибокого навчання для прогнозування значення функції цілі використовувались повнозв'язані нейронні мережі, згорткові та рекурентні нейронні мережі, а також були розроблені та використані алгоритми автоматичного підбору гіперпараметрів розроблених моделей, як наведено нижче.

```
def build_model(hp):
    model = Sequential()
    model.add(Dense(units=hp.Int('units_input',
                                 min_value=128,     # мін. кіл нейронів - 128
                                 max_value=1024,    # макс  - 1024
                                 step=32),
                    input_dim=330,
                    activation='relu'))

    model.add(Dense(units=hp.Int('units_hidden',
                                 min_value=128,
                                 max_value=600,
                                 step=32),
                    activation='relu'))


    model.add(Dense(units=hp.Int('units_hidden',
                                 min_value=16,
                                 max_value=128,
                                 step=32),
                    activation='relu'))
    model.add(Dense(1))
    model.compile(
        optimizer=hp.Choice('optimizer', values=['adam']),
        loss='mse',
        metrics='mae')
    return model
```

Результатом виконання функції автоматичного підбору гіперпараметрів для повнозв'язної мережі з двома прихованими шарами є декілька варіантів, які можна використати для використання нейромережі.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 512)               169472

 dense_1 (Dense)             (None, 512)               262656

 dense_2 (Dense)             (None, 512)               262656

 dense_3 (Dense)             (None, 1)                 513

=================================================================
Total params: 695,297
Trainable params: 695,297
Non-trainable params: 0
_________________________________________________________________
43/43 [==============================] - 0s 3ms/step - loss: 0.0095 - mae: 0.0513

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 928)               307168

 dense_1 (Dense)             (None, 576)               535104

 dense_2 (Dense)             (None, 576)               332352

 dense_3 (Dense)             (None, 1)                 577

=================================================================
Total params: 1,175,201
Trainable params: 1,175,201
Non-trainable params: 0
_________________________________________________________________
43/43 [==============================] - 0s 3ms/step - loss: 0.0100 - mae: 0.0553
```

На рисунку 3.6 відображено залежність передбачуваних повнозв’язною нейромережою даних від реальних даних для тестової вибірки.

Спостерігається гарна відповідність отриманих даних у порівнянні з реальними даними тестової вибірки. Це підтверджується графіком залежності (рисунок 3.7) значення епістаза для значень тестової вибірки і передбачених алгоритмом.

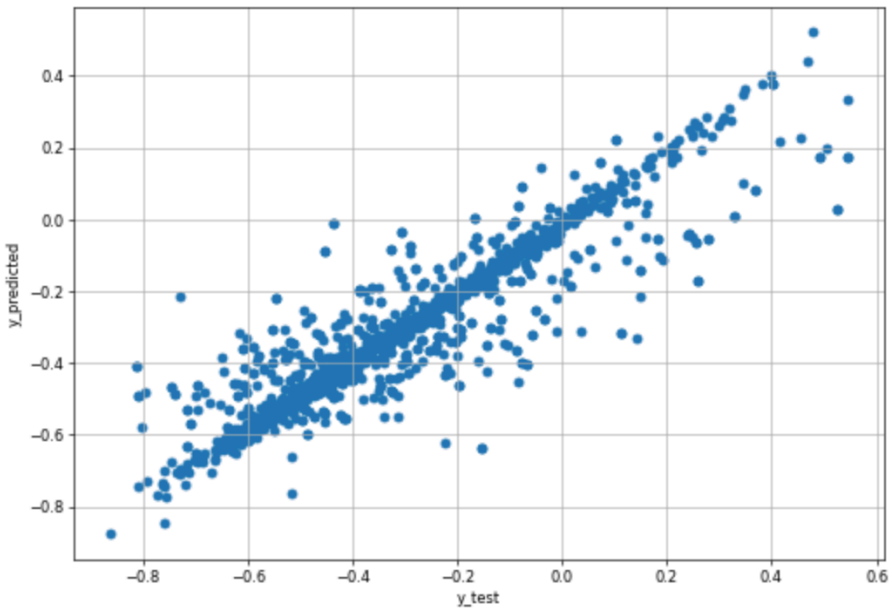

Рисунок 3.6 – Залежність передбачуваних повнозв’язною нейромережою даних від реальних даних для тестової вибірки

У таблиці 1 наведено результати деяких моделей машинного навчання інформаційної технології, що демонструють точність використаних алгоритмів.

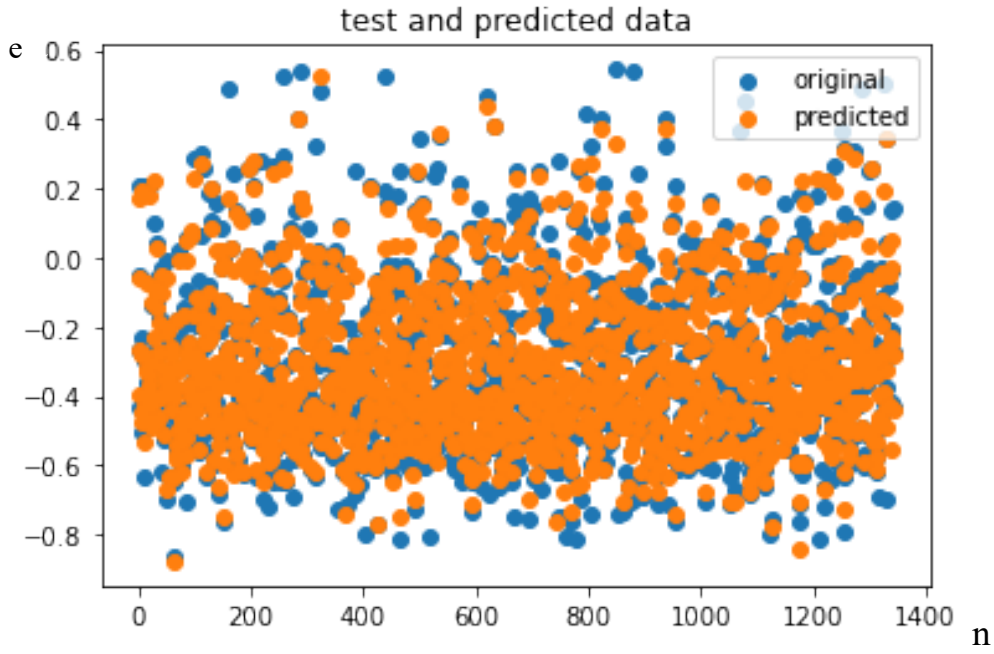

Рисунок 3.7 – Передбачувані та реальні дані тестової вибірки, отримані повнозв’язною нейромережою

Архітектура рекурентної нейронної мережі представлена в наступному програмному коді:

```
model = keras.models.Sequential([
    keras.layers.SimpleRNN(20, return_sequences=True, input_shape=[None, 1]),
    keras.layers.SimpleRNN(20, return_sequences=True),
    keras.layers.SimpleRNN(1)
])

model.compile(loss="mse", optimizer="adam")
history = model.fit(X_train, y_train, epochs=20,
                    validation_data=(X_valid, y_valid))
```

Таблиця 1 – Результати якості моделей машинного навчання

	Моделі машинного навчання інформаційної технології	Коеф. детермінації, навчальна вибірка	Коеф. детермінації, тренувальна вибірка	MAE	Коеф. кореляції Пірсона/ Спірмана
Алгоритми машинного навчання з учителем					
1	Random Forest Regressor	0.90	0.67	0.01	0.94/0.93
2	Decision Tree Regressor	0.85	0.80	0.04	0.92/0.91
3	Gradient Boosting Regressor	0.87	0.69	0.09	0.93/0.92

Використання нейромереж зі складними архітектурами (глибоке навчання) дають наступні результати:

- повнозв’язна нейронна мережа (MAE=0.06, коефіцієнт кореляції Пірсона/Спірмана 0.91/0.90),
- згорткова нейронна мережа (MAE=0.03, коефіцієнт кореляції Пірсона/Спірмана 0.92/0.90),
- рекурентна нейронна мережа (MAE=0.06, коефіцієнт кореляції Пірсона/Спірмана 0.93/0.89).

Отримані дані відповідають відомим результатам для біомедичних зразків протеїнів інших типів, що складає значення коефіцієнта кореляції Пірсона або Спірмана між тестовими і передбачуваними даними, які становлять значення 0,5 – 0,8.

ВИСНОВКИ

В результаті виконання магістерської кваліфікаційної роботи було проведено аналіз літературних джерел і алгоритмів машинного навчання, що використовуються в інформаційних технологіях в біоінженерії. Також здійснено огляд програмних засобів реалізації поставленої задачі. Для реалізації поставленої задачі обрано мову програмування Python, та бібліотеки Pandasб Mathplotlib, Seaborn, TensorFlow. Спроектовано інформаційну технологію машинного навчання в задачах біоінженерії. Підготовлено вхідні дані та розроблені алгоритми машинного навчання.

Проектування і програмна реалізація інформаційної технології включала розробку моделей машинного навчання Random Forest, DecisionTreeRegressor, Gradient Boosting Regressor, XGBRegressor, SVR. Також використовувались ансамблеві алгоритми Bagging і Boosting на їх основі.

В якості алгоритмів глибокого навчання для прогнозування значення функції цілі використовувались повнозв’язані нейронні мережі, згорткові та рекурентні нейронні мережі, для яких були розроблені та використані алгоритми автоматичного підбору гіперпараметрів розроблених моделей.

Отримані результати корелюють з відомими подібними даними, що говорить про високу якість запропонованої інформаційної моделі машинного навчання.

СПИСОК ЛІТЕРАТУРИ

1. Rollins N. J., Brock K. P., Poelwijk F. J., Stiffler M. A., Gauthier N. P., Sander C., and Marks D. S. Inferring protein 3D structure from deep mutation scans, Nature Genetics 51, 2020. – PP.1170 – 1176.

2. Singh V. K., Maurya N. Sh., Mani A., Yadava R. Sh. Machine learning method using position-specific mutation based classification outperforms one hot coding for disease severity prediction in haemophilia ‘A’, Genomics 112, 2020. – PP.5122-5128.

3. Gelman S., Fahlberg S., Heinzelman P., Romero P. A., and Gitter A., Neural networks to learn protein sequence–function relationships from deep mutational scanning data. PNAS 118, 2021. – PP.48 – 60.

4. Freschlin Ch. R., Fahlberg S. A, Romero Ph. A Machine learning to navigate fitness landscapes for protein engineering, Current Opinion in Biotechnology 75, 2022 – PP.112-124.

5. Feng S., Zhou H., Dong H. Using deep neural network with small dataset to predict material defects, Materials and Design 162 , 2019. – PP.300–310.

6. Cai Y., Wang J., Deng L., SDN2GO: An Integrated Deep Learning Model for Protein Function Prediction, Front. Bioeng. Biotechnol. 8, 2020 . – PP.234-245.

7. Xu Y., Verma D., Sheridan R. P., Liaw A., Ma J., Marshall N. M., McIntosh J., Sherer E. C., Svetnik V., and Johnston J. M. Deep Dive into Machine Learning Models for Protein Engineering, J. Chem. Inf. Model. 60, 2020. – PP.2773−2790.

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

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

10. Hastie T., Tibshirani R., Friedman J. The Elements of Statistical Learning Data Mining, Inference, and Prediction. - Springer. – 2019. - 765 p.
11. McKinney W. Python for Data Analysis. – Sebastopol: O'Reilly Media, 2019. – 470 p.
12. 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.
13. Marskand S. Machine Learning: An Algorithmic Perspective. – New York: Taylor & Francis Group, 2019. – 452 p.
14. 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.
15. Bowles M. Machine Learning in Python. – Indianapolis: John Wiley & Sons, 2017. - 361 p.
16. Unpingco J. Python for Probability, Statistics, and Machine Learning. -San Diego: Springer, 2018. – 288 p.
17. Duchesnay E., LöfstedtT. Statistics and Machine Learning in Python. – New York: Springer, 2017. – 169 p.
18. Swamynathan M. Mastering Maching Learning with Python in Six Steps. A Practical Implementation Guide to Predictive Data Analytics Using Python. – Karnataka: Apress, 2017. – 374p.
19. 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
import mglearn
from sklearn import linear_model
import pandas as pd
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 numpy as np
import seaborn as sns
from scipy import stats

import sys
if not sys.warnoptions:
    import warnings
    warnings.simplefilter("ignore")

dk = pd.read_csv("Full_dataset.tsv", on_bad_lines='skip', delimiter = "\t",
low_memory=False, encoding = 'utf-8')
dk=dk.drop(['tag'],axis=1)
dk=dk.drop(['cells_rep1'],axis=1)
dk=dk.drop(['cells_rep2'],axis=1)
dk['mean'] = (dk['fitness_rep1']+dk['fitness_rep2'])/2
dk['n_aa_substitutions_syn'] = dk['n_aa_substitutions_syn'].astype(int)
```

```
dk=dk.drop(['fitness_rep1'],axis=1)
dk=dk.drop(['fitness_rep2'],axis=1)
dk.reset_index(drop=True, inplace=True)
df= dk.query('n_aa_substitutions_syn==2')
df=df.dropna()

#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'
))
dd=df.drop(['n_aa_substitutions_syn'],axis=1)
dd['aa_mutation1'] = dd['aa_mutation_syn'].apply(lambda x: x.split(' ')[0])
dd['aa_mutation2'] = dd['aa_mutation_syn'].apply(lambda x: x.split(' ')[1])
dd['wt_aa1'] = dd['aa_mutation1'].apply(lambda x: x[0:1])
dd['wt_aa2'] = dd['aa_mutation2'].apply(lambda x: x[0:1])
dd['mutated_aa1'] = dd['aa_mutation1'].apply(lambda x: x[-1:])
dd['mutated_aa2'] = dd['aa_mutation2'].apply(lambda x: x[-1:])
dd['pos1'] = dd['aa_mutation1'].apply(lambda x: x[1:-1])
dd['pos1'] = dd['pos1'].astype(int)
dd['pos1'] = dd['pos1']-136
dd['pos2'] = dd['aa_mutation2'].apply(lambda x: x[1:-1])
dd['pos2'] = dd['pos2'].astype(int)
dd['pos2'] = dd['pos2']-136
```

```
dd = dd.query('pos1 >0 and pos1<155')
dd = dd.query('pos2 >0 and pos1<155')
dd.reset_index(drop=True, inplace=True)

dr = pd.read_csv("Myo_epi_3d.tsv", on_bad_lines='skip',  delimiter = "\t",
header=0, low_memory=False,encoding ='unicode_escape')
dr.reset_index(drop=True, inplace=True)
for i in tqdm_notebook(dd.index):
    for j in dr.index:
        if dd.at[i,'aa_mutation_syn'] == dr.at[j,'aa_mutation_syn']:
            dd.at[i,'distance'] = dr.at[j,'distance']
            dd.at[i,'epistasis'] = dr.at[j,'epistasis']
            #dd.at[i,'delta_mean'] = dr.at[j,'delta_mean']

dd=dd.dropna()
dd=dd.query('distance<10')
dd.reset_index(drop=True, inplace=True)
#H-bond S ot T
dd = dd.query('mutated_aa1=="S" or mutated_aa1=="T" and mutated_aa2=="S" or
mutated_aa2=="T" '
for i in tqdm_notebook(dd.index):
    for j in dn.index:
        if dd.at[i,'mutated_aa1'] == dn.at[j,'AA']:
            dd.at[i,'bulkiness_aa1'] = dn.at[j,'Bulkiness']
            dd.at[i,'polarity_aa1'] = dn.at[j,'Polarity']
            dd.at[i,'hydrophobicity_aa1'] = dn.at[j,'Hydrophobicity1']
            dd.at[i,'mutability_aa1'] = dn.at[j,'Mutability']
            dd.at[i,'Pol_aa1'] = dn.at[j,'Pol']
```

```
    if dd.at[i,'mutated_aa2'] == dn.at[j,'AA']:
            dd.at[i,'bulkiness_aa2'] = dn.at[j,'Bulkiness']
            dd.at[i,'polarity_aa2'] = dn.at[j,'Polarity']
            dd.at[i,'hydrophobicity_aa2'] = dn.at[j,'Hydrophobicity1']
            dd.at[i,'mutability_aa2'] = dn.at[j,'Mutability']
            dd.at[i,'Pol_aa2'] = dn.at[j,'Pol']
        if dd.at[i,'wt_aa1'] == dn.at[j,'AA']:
            dd.at[i,'bulkiness_wt1'] = dn.at[j,'Bulkiness']
            dd.at[i,'polarity_wt1'] = dn.at[j,'Polarity']
            dd.at[i,'hydrophobicity_wt1'] = dn.at[j,'Hydrophobicity1']
            dd.at[i,'mutability_wt1'] = dn.at[j,'Mutability']
            dd.at[i,'Pol_wt1'] = dn.at[j,'Pol']
        if dd.at[i,'wt_aa2'] == dn.at[j,'AA']:
            dd.at[i,'bulkiness_wt2'] = dn.at[j,'Bulkiness']
            dd.at[i,'polarity_wt2'] = dn.at[j,'Polarity']
            dd.at[i,'hydrophobicity_wt2'] = dn.at[j,'Hydrophobicity1']
            dd.at[i,'mutability_wt2'] = dn.at[j,'Mutability']
            dd.at[i,'Pol_wt2'] = dn.at[j,'Pol']
#mut1 - mut2
dd['delta_bulkiness'] = abs(dd['bulkiness_aa2'] - dd['bulkiness_aa1'])#
dd['delta_polarity'] = abs(dd['polarity_aa2'] - dd['polarity_aa1'])#
dd['delta_hydrophobicity'] = abs(dd['hydrophobicity_aa2'] -
dd['hydrophobicity_aa1'])#
dd['delta_mutability'] = abs(dd['mutability_aa2'] - dd['mutability_aa1'])#
#wt1 - mut1
dd['delta_bulkiness_wt_aa1'] = dd['bulkiness_aa1'] - dd['bulkiness_wt1']#abs()
dd['delta_polarity_wt_aa1'] = dd['polarity_aa1'] - dd['polarity_wt1']#abs()
dd['delta_hydrophobicity_wt_aa1'] = dd['hydrophobicity_aa1'] -
dd['hydrophobicity_wt1']#abs()
```

```
dd['delta_mutability_wt_aa1'] = dd['mutability_aa1'] - dd['mutability_wt1']#abs()
#wt2 - mut2
dd['delta_bulkiness_wt_aa2'] = dd['bulkiness_aa2'] - dd['bulkiness_wt2']#abs()
dd['delta_polarity_wt_aa2'] = dd['polarity_aa2'] - dd['polarity_wt2']#abs()
dd['delta_hydrophobicity_wt_aa2'] = dd['hydrophobicity_aa2'] -
dd['hydrophobicity_wt2']#abs()
dd['delta_mutability_wt_aa2'] = dd['mutability_aa2'] - dd['mutability_wt2']#abs()

dd = dd.drop(['aa_mutation_syn'], axis=1)
dd['aa_mutation_1'] = dd['wt_aa1']+dd['pos1'].astype(str)+dd['mutated_aa1']
dd['aa_mutation_2'] = dd['wt_aa2']+dd['pos2'].astype(str)+dd['mutated_aa2']
dd = dd.drop(['aa_mutation1'], axis=1)
dd = dd.drop(['aa_mutation2'], axis=1)
dd = dd.drop(['wt_aa1'], axis=1)
dd = dd.drop(['wt_aa2'], axis=1)
dff=dd.groupby("aa_mutation_syn")["mean"].mean()

dk=dk.query('n_aa_substitutions_syn ==1')
dk=dk.dropna()
dk['wt_aa'] = dk['aa_mutation_syn'].apply(lambda x: x[0:1])
dk['mutated_aa'] = dk['aa_mutation_syn'].apply(lambda x: x[-2:])
dk['pos'] = dk['aa_mutation_syn'].apply(lambda x: x[1:-2])
dk['pos1'] = dk['pos'].astype(int)
dk['pos1'] = dk['pos1']-136
dk = dk.query('pos1 >0 and pos1<155')
dk['mutat'] = dk['wt_aa']+dk['pos1'].astype(str)+dk['mutated_aa']
dk.reset_index(drop=True, inplace=True)
dk['mutat']=dk['mutat'].apply(lambda x: x.strip())
```

```
dd = dd.drop(['aa_mutation_syn'], axis=1)
dd = dd.drop(['mutated_aa1'], axis=1)
dd = dd.drop(['mutated_aa2'], axis=1)
dd = dd.drop(['pos1'], axis=1)
dd = dd.drop(['pos2'], axis=1)
dd = dd.drop(['bulkiness_aa1'], axis=1)
dd = dd.drop(['bulkiness_aa2'], axis=1)
dd = dd.drop(['polarity_aa1'], axis=1)
dd = dd.drop(['polarity_aa2'], axis=1)
dd = dd.drop(['hydrophobicity_aa1'], axis=1)
dd = dd.drop(['hydrophobicity_aa2'], axis=1)
dd = dd.drop(['mutability_aa1'], axis=1)
dd = dd.drop(['mutability_aa2'], axis=1)

for i in tqdm_notebook(dd.index):
    for j in dk.index:
        if dd.at[i,'aa_mutation_1'] == dk.at[j,'mutat']:
            dd.at[i,'mean_aa1'] = dk.at[j,'mean']
        if dd.at[i,'aa_mutation_2'] == dk.at[j,'mutat']:
            dd.at[i,'mean_aa2'] = dk.at[j,'mean']
dd=dd.dropna()

de = pd.DataFrame()
de['mean'] = dd['mean']
de['delta_bulkiness'] = dd['delta_bulkiness']
de['delta_polarity'] = dd['delta_polarity']
de['delta_mutability'] = dd['delta_mutability']
de['delta_hydrophobicity'] = dd['delta_hydrophobicity']
de['mean_aa1'] = dd['mean_aa1']
```

```
de['mean_aa2'] = dd['mean_aa2']
de['mutated_aa1'] = dd['mutated_aa1']
de['mutated_aa2'] = dd['mutated_aa2']
de['pos1'] = dd['pos1'].astype('str')
de['pos2'] = dd['pos2'].astype('str')
de['distance'] = dd['distance']
de=de.query('distance<10')
de['epistasis'] = dd['epistasis']

de['delta_bulkiness_wt_aa1'] = dd['delta_bulkiness_wt_aa1']
de['delta_polarity_wt_aa1'] = dd['delta_polarity_wt_aa1']
de['delta_mutability_wt_aa1'] = dd['delta_mutability_wt_aa1']
de['delta_hydrophobicity_wt_aa1'] = dd['delta_hydrophobicity_wt_aa1']
de['delta_bulkiness_wt_aa2'] = dd['delta_bulkiness_wt_aa2']
de['delta_polarity_wt_aa2'] = dd['delta_polarity_wt_aa2']
de['delta_mutability_wt_aa2'] = dd['delta_mutability_wt_aa2']
de['delta_hydrophobicity_wt_aa2'] = dd['delta_hydrophobicity_wt_aa2']
de['aa_mutation_1'] = dd['aa_mutation_1']
de['aa_mutation_2'] = dd['aa_mutation_2']
de['epistasis'] = de['mean'] - de['mean_aa1'] -de['mean_aa2']
de=de.drop('mean', axis=1)
de=de.append(de)
de=de.drop('mean_aa1', axis=1)
de=de.drop('mean_aa2', axis=1)
hist_mean = de.hist(column='delta_polarity',bins=100)
hist_pos1 = de.hist(column='epistasis',bins=10)
de.reset_index(drop=True, inplace=True)
plt.figure(figsize = (16,7))
sns.heatmap(de.corr(), annot=True,fmt='.1g', cmap="PuBu")
```

```
sns.pairplot(de)

dd_bin = pd.get_dummies(de)

#future selection
# evaluation of a model using features chosen with mutual information
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import mutual_info_regression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error

def scalling(X_train,X_test):
    scaler = StandardScaler()
    scaler.fit(X_train)
    X_train_scaled = scaler.transform(X_train)
    X_test_scaled = scaler.transform(X_test)
    return X_train_scaled, X_test_scaled

def select_features(X_train, y_train, X_test):

        fs = SelectKBest(score_func=mutual_info_regression,k=10)
        fs.fit(X_train, y_train)
        X_train_fs = fs.transform(X_train)
        X_test_fs = fs.transform(X_test)
        return X_train_fs, X_test_fs, fs
X_train, X_test, y_train, y_test = train_test_split(dd_bin.drop(['mean'],axis=1),
                                        dd_bin['mean'],random_state=1)
```

```
X_train_scaled, X_test_scaled = scalling(X_train,X_test)
# feature selection
X_train_fs, X_test_fs, fs = select_features(X_train_scaled, y_train, X_test_scaled)
# fit the model
model= RandomForestRegressor()
model.fit(X_train_fs, y_train)
# evaluate the model
yhat = model.predict(X_test_fs)
# evaluate predictions
mae = mean_absolute_error(y_test, yhat)
print('MAE: %.3f' % mae)

for i in range(len(fs.scores_)):
	print('Feature %d: %f' % (i, fs.scores_[i]))
# plot the scores
pyplot.bar([i for i in range(len(fs.scores_))], fs.scores_)
pyplot.show()

#DecisionTreeRegressor
#scaling
X_train, X_test, y_train, y_test = train_test_split(dd_bin.drop(['mean'],axis=1),
dd_bin['mean'],
                                                    test_size=0.2, random_state=0)

tree_reg = DecisionTreeRegressor()
sscaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
plt.scatter(x_ax, y_test, label="original")
  tree_reg.fit(X_train_scaled, y_train)
  y_predict = tree_reg.predict(X_test_scaled)
  # Evaluate the model
  mae = np.mean(abs(y_predict - y_test))
  print('DecisionTreeRegressor Performance on the test set: MAE = %0.4f' % mae)
  print('mean squared error',mean_squared_error(y_test,y_predict))
  print(r2_score(y_test, y_predict))
  dd_show = pd.DataFrame()
  dd_show['y_test'] = y_test
  dd_show['y_predict'] = y_predict
  #dd_show
  print(dd_show.corr(method='pearson'))

  #RandomForestRegressor
  from sklearn.ensemble import RandomForestRegressor
  random_forest = RandomForestRegressor()
  random_forest.fit(X_train, y_train)
  y_predict = random_forest.predict(X_test)
  # Evaluate the model
  mae = np.mean(abs(y_predict - y_test))
  print('Random Forest Performance on the test set: MAE = %0.4f' % mae)
  print('mean squared error',mean_squared_error(y_test,y_predict))
  print('Coefficient of determination',r2_score(y_test, y_predict))
  dd_show = pd.DataFrame()
  dd_show['y_test'] = y_test
  dd_show['y_predict'] = y_predict
  #dd_show
  plt.rcParams.update({'font.size': 10})
  x_ax = range(len(X_test))
```

```
plt.scatter(x_ax, y_predict, label="predicted")
plt.title("test and predicted data")
plt.legend()
plt.show()
dd_show = pd.DataFrame()
dd_show['y_test'] = y_test
dd_show['y_predict'] = y_predict
print(dd_show.corr(method='pearson'))
print(dd_show.corr(method='spearman'))

# DecisionTreeRegressor
from sklearn.tree import DecisionTreeRegressor
X_train, X_test, y_train, y_test = train_test_split(dd_bin.drop(['epistasis'],axis=1),
dd_bin['epistasis'],test_size=0.2, random_state=40)

tree_reg = DecisionTreeRegressor()

scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
tree_reg.fit(X_train_scaled, y_train)
y_predict = tree_reg.predict(X_test_scaled)
y_predict_tr= tree_reg.predict(X_train_scaled)

# Evaluate the model
mae = np.mean(abs(y_predict - y_test))

dd_show = pd.DataFrame()
```

```
dd_show['y_test'] = y_test
dd_show['y_predict'] = y_predict
print(dd_show.corr(method='pearson'))

print(dd_show.corr(method='spearman'))
print('DecisionTreeRegressor Performance on the test set: MAE = %0.4f' % mae)
print('mean squared error',mean_squared_error(y_test,y_predict))
print('r2',np.sqrt(mean_squared_error(y_test,y_predict)))
print('Accuracy R-squared', metrics.r2_score(y_test,y_predict))
print('Accuracy R-squared', metrics.r2_score(y_train,y_predict_tr))

# GradientBoostingRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import cross_val_score
x = dd_bin.drop(['epistasis'],axis=1)
y = dd_bin['epistasis']
X_train, X_test, y_train, y_test = train_test_split(x, y,random_state=30, test_size =
0.2)
# Create the model
gradient_boosted = GradientBoostingRegressor(random_state=5)
# cross-validation
scores = cross_val_score(gradient_boosted,dd_bin.drop(['epistasis'],axis=1),
dd_bin['epistasis'], cv=5)
print("accuracy on cross-validation: {}".format(scores.mean()))

#scaling
scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
```

```
X_test_scaled = scaler.transform(X_test)
gradient_boosted.fit(X_train_scaled, y_train)
y_predict = gradient_boosted.predict(X_test_scaled)

# Evaluate the model
mae = np.mean(abs(y_predict - y_test))

print('Gradient Boosted Performance on the test set: MAE = %0.4f' % mae)
print('mean squared error',mean_squared_error(y_test,y_predict))

print('Accuracy r2 =',metrics.r2_score(y_test, y_predict))
dd_show = pd.DataFrame()
dd_show['y_test'] = y_test
dd_show['y_predict'] = y_predict

#enssembles
from sklearn.base import TransformerMixin
from sklearn.datasets import make_regression
from sklearn.pipeline import Pipeline, FeatureUnion
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.linear_model import LinearRegression, Ridge
class RidgeTransformer(Ridge, TransformerMixin):
    def transform(self, X, *_):
        return self.predict(X).reshape(len(X), -1)
class RandomForestTransformer(RandomForestRegressor, TransformerMixin):
    def transform(self, X, *_):
```

```
        return self.predict(X).reshape(len(X), -1)

class KNeighborsTransformer(KNeighborsRegressor, TransformerMixin):

    def transform(self, X, *_):
        return self.predict(X).reshape(len(X), -1)
def build_model():
    ridge_transformer = Pipeline(steps=[
        ('scaler', StandardScaler()),
        ('poly_feats', PolynomialFeatures()),
        ('ridge', RidgeTransformer())
    ])

    pred_union = FeatureUnion(
        transformer_list=[
            ('ridge', ridge_transformer),
            ('rand_forest', RandomForestTransformer()),
            ('knn', KNeighborsTransformer())
        ],
        n_jobs=2
    )
    model = Pipeline(steps=[
        ('pred_union', pred_union),
        ('lin_regr', LinearRegression())
    ])
    return model
model = build_model()
X, y = make_regression(n_features=10)
```

```
X_train, X_test, y_train, y_test = train_test_split(dd_bin.drop(['mean'],axis=1),
dd_bin['mean'], test_size=0.2,random_state=42)

model.fit(X_train, y_train)
score = model.score(X_test, y_test)
#print('Score:', score)
y_predict = model.predict(X_test)

dd_show = pd.DataFrame()
dd_show['y_test'] = y_test
dd_show['y_predict'] = y_predict
print(dd_show.corr(method='pearson'))

# Evaluate the model
mae = np.mean(abs(y_predict - y_test))
print("Accuracy on train data set:{:.2f}".format(model.score(X_train, y_train)))
print("Accuracy on test data set: {:.2f}".format(model.score(X_test, y_test)))
print('GaussianProcessRegressor Performance on the test set: MAE = %0.4f' %
mae)
print('mean squared error',mean_squared_error(y_test,y_predict))
print('r2',np.sqrt(mean_squared_error(y_test,y_predict)))
print(r2_score(y_test, y_predict))
x_ax = range(len(X_test))
plt.figure(figsize=(50, 10), dpi=80)
plt.plot(x_ax, y_test, label="original",linewidth=4)
plt.plot(x_ax, y_predict, label="predicted",linewidth=4)
plt.title("test and predicted data, x = number of rows in a test data")
plt.legend()
plt.show()
```

```
#tuner
from kerastuner import RandomSearch
X_train, X_test, y_train, y_test = train_test_split( dd_bin.drop(['epistasis'],axis=1),
dd_bin['epistasis'],random_state=30)

scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

def build_model(hp):
   model = Sequential()
   model.add(Dense(units=hp.Int('units_input',
                        min_value=128,    # мін. кіл нейронів - 128
                        max_value=1024,   # макс  - 1024
                        step=32),
               input_dim=330,
               activation='relu'))

   model.add(Dense(units=hp.Int('units_hidden',
                        min_value=128,
                        max_value=600,
                        step=32),
               activation='relu'))
```

```
    model.add(Dense(units=hp.Int('units_hidden',
                                 min_value=16,
                                 max_value=128,
                                 step=32),
                    activation='relu'))
    model.add(Dense(1))
    model.compile(
        optimizer=hp.Choice('optimizer', values=['adam']),
        loss='mse',
        metrics='mae')
    return model
tuner = RandomSearch(
    build_model,
    objective='mae',

    max_trials=10,
    directory='test_diiiresr'
    )
tuner.search_space_summary()
tuner.search(X_train_scaled,
             y_train,
             batch_size=32,
             epochs=50,
             validation_split=0.2,
             )
#FCNN
from keras.callbacks import EarlyStopping
import keras as k
```

```
X_train, X_test, y_train, y_test = train_test_split( dd_bin.drop(['epistasis'],axis=1),
dd_bin['epistasis'],random_state=42)
scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
model = k.Sequential()
model.add(k.layers.Dense(units=600, activation="relu"))
model.add(k.layers.Dense(units=124, activation="relu"))
model.add(k.layers.Dense(units=64, activation="relu"))
model.add(k.layers.Dense(units=1))
model.compile(loss="mse", optimizer="adam", metrics = 'mae')
model.save_weights("weigts.h5")
fit_res = model.fit(X_train_scaled,  y_train, epochs = 100,batch_size=16,
validation_split=0.2,
                callbacks=EarlyStopping(monitor='val_loss', mode='min', verbose=1,
patience=50))
mse, mae = model.evaluate(X_test_scaled, y_test, verbose = 0)
print ('mae=',mae)
print ('mse=',mse)
x_ax = range(len(X_test_scaled))
y_pred = model.predict(X_test_scaled)
y_pred_train = model.predict(X_train_scaled)
dd_show = pd.DataFrame()
dd_show['y_test'] = y_test
dd_show['y_predict'] = y_pred

#dd_show
print('R2score on the Test set', metrics.r2_score (y_test, y_pred))
```

```
print('R2score on the Train set', metrics.r2_score (y_train, y_pred_train))
dd_show = pd.DataFrame()
dd_show['y_test'] = y_test
dd_show['y_predict'] = y_pred
print(dd_show.corr(method='pearson'))
print(dd_show.corr(method='spearman'))
plt.figure(figsize=(10, 7), dpi=60)
plt.scatter(y_test, y_pred)
plt.xlabel("y_test")
plt.ylabel("y_predicted")
plt.grid(True)
```