МІНІСТЕРСТВО ОСВІТИ I НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ   
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ БІОМЕДИЧНОЇ ІНЖЕНЕРІЇ

КАФЕДРА БІОМЕДИЧНОЇ КІБЕРНЕТИКИ

**РОЗРАХУНКОВО-ГРАФІЧНА РОБОТА**

з дисципліни «Нечіткі моделі в медицині»

на тему: «Класифікатор у вигляді багатошарового персептрону, сформованого на основі методу пошуку у вигляді генетичного алгоритму.»

Варіант №13

**Виконав:**

студент гр. БС-81

Погребенко В.О.

**Перевірив:**

доц. каф. БМК

Добровська Л.М.

Зараховано від \_\_\_.\_\_\_.\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис викладача)

Київ-2020

ЗМІСТ

[ВСТУП 3](#_Toc61029507)

[АНОТАЦІЯ 4](#_Toc61029508)

[ОСНОВНА ЧАСТИНА 7](#_Toc61029509)

[Методи виконання завдання 7](#_Toc61029510)

[Розділ 1. Дані для завдання 8](#_Toc61029511)

[Розділ 2. Етапи виконання завдання 10](#_Toc61029512)

[2.1 Створення багатошарового персептрону. 10](#_Toc61029513)

[2.2 Реалізація генетичного алгоритму 10](#_Toc61029514)

[Розділ 3. Остаточний результат 12](#_Toc61029515)

[ВИСНОВКИ 15](#_Toc61029516)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 16](#_Toc61029517)

[ДОДАТОК А 18](#_Toc61029518)

[ДОДАТОК Б 20](#_Toc61029519)

# ВСТУП

Актуальність розрахунково-графічної роботи роботи полягає у створенні класифікатору у вигляді багатошарового персептрону, сформованого на основі методу пошуку у вигляді генетичного алгоритму. Отриманний багатошаровий персептрон будє за допомогою зібранних даних з пацієнта таких як: нахил тазу, поперековий кут лордозу, етс будє визначати, чи є аномалії хребету у пацієнта.

Так як біль у попереку є надзвичайно поширеним явищем, симптоми та вираженість болю в попереку дуже різняться. Вони можуть бути спричинені різноманітними проблемами з будь-якими частинами складної, взаємопов’язаної мережі спинномозкових м’язів, нервів, кісток, дисків або сухожиль в поперековому відділі хребта. [1]

Таким чином, хоча обстеження можуть бути у нормі згідно з прийнятими значеннями, існує можливість що проблеми викликані комплексом змін, що не виходять за границі норми, та які важко діагностувати звичайній людині. Тому, классифікація аномальності результатів завдяки завдяки MLP є актуальним та важливим завданням для дослідження пацієнтів. [2]

Але для того, щоб сформувати оптимальний MLP, необхідно знайти найкращу кількість слоїв прихованих нейронів, функцію активації, етс, що майже неможливо зробити «вручну». Тому для пошуку формування оптимального MLP будє використаний генетичний алгоритм, що дозволить знайти рішення для цієї задачі.

У результаті, буде сформований MLP на основі методу пошуку у вигляді генетичного алгоритму, що дозволить класифікувати результати досліджень стану хребта паціентів.

# АНОТАЦІЯ

Розрахунково-графічну роботу виконав Погребенко Василь Олександрович, студент 3 курсу, групи БС-81, кафедри біомедичної кібернетики, факультету біомедичної інженерії НТУУ «КПІ ім. Ігоря Сікорського».

Тема розрахунково-графічної роботи: Класифікатор у вигляді багатошарового персептрону, сформованого на основі методу пошуку у вигляді генетичного алгоритму.

Структура і обсяг роботи: розрахунково-графічна робота складається із вступу, основної частини, висновків, списку використаної літератури із трьох джерел, та двох додатків. Загальний обсяг розрахунково-графічної роботи становить 31 сторінку, основного тексту (без додатків) – 17 сторінок, іллюстрацій – 8, таблиць - 1.

**АННОТАЦИЯ**

Расчетно-графическую работу выполнил Погребенко Василий Александрович, студент 3 курса, группы БС-81, кафедры биомедицинской кибернетики, факультета биомедицинской инженерии НТУУ «КПИ им. Игоря Сикорского».

Тема расчетно-графической работы: Классификатор в виде многослойного персептрона, сформированного на основе метода поиска в виде генетического алгоритма.

Структура и объем работы: расчетно-графическая работа состоит из введения, основной части, заключения, списка использованной литературы из трех источников, и двух приложений. Общий объем расчетно-графической работы составляет 31 страницу, основного текста (без приложений) - 17 страниц, иллюстраций – 8, таблиц - 1.

**ABSTRACT**

The calculation and graphic work was performed by Pohrebenko Vasyl, 3rd year student, group BS-81, Department of Biomedical Cybernetics, Faculty of Biomedical Engineering, NTUU “KPI named after Igor Sikorsky. "

The topic of calculation and graphic work: Classifier in the form of a multilayer perceptron, formed on the basis of the search method in the form of a genetic algorithm.

Structure and scope of work: calculation and graphic work consists of the introduction, the main part, conclusion, the list of references from three sources, and two applications. The total amount of calculation and graphic work consists of 31 pages of the main text (without attachments) - 17 pages, illustrations – 8, tables - 1.

# ОСНОВНА ЧАСТИНА

## Методи виконання завдання

У завданні використовується багатошаровий персептрон (MLP) (використовується бібліотека з python sklearn), та генетичний алгоритм

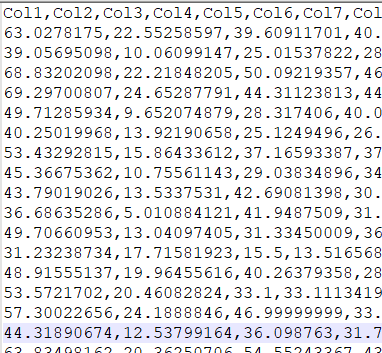
Багатошаровий персептрон (MLP) - це вид нейронної мережі, що складається щонайменше з трьох шарів вузлів: вхідного шару, прихованого шару та вихідного шару. За винятком вхідних вузлів, кожен вузол є нейроном, який використовує нелінійну функцію активації. MLP використовує контрольовану техніку навчання, яка називається зворотним розповсюдженням для навчання (backpropagation). Його багатошаровість і нелінійна активація відрізняють MLP від лінійного персептрона. Він може розрізняти дані, які не можна лінійно розділити.

Генетичний алгоритм - це евристичний алгоритм пошуку що принципом дії нагадує процес природного Генетичні алгоритми зазвичай використовуються для рішення задач щодо оптимізації та пошуку, що використовує мутації, кросовери, etc

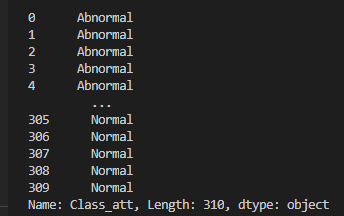
Виконання завдання поділяється на 3 розділи.

## Розділ 1. Дані для завдання

Для завдання використовується «Lower Back Pain Symptoms Dataset», що містить данні про дані досліджень пацієнтів їх зібрані фізичні дані / дані стану хребта. Повний сет даних приведений у додатку А.



*Рис. 1. Вміст файлу з зібранними даними*



*Рис. 2. Колонка, що описує, чи є дані аномальними.*

|  |  |
| --- | --- |
| **Attribute name** | **type** |
| Col1 | pelvic\_incidence |
| Col2 | pelvic\_tilt |
| Col3 | lumbar\_lordosis\_angle |
| Col4 | sacral\_slope |
| Col5 | pelvic\_radius |
| Col6 | degree\_spondylolisthesis |
| Col7 | pelvic\_slope |
| Col8 | Direct\_tilt |
| Col9 | thoracic\_slope |
| Col10 | cervical\_tilt |
| Col11 | sacrum\_angle |
| Col12 | scoliosis\_slope |
| Class\_att | Attribute Class |
| Unnamed | dummy Column |

*Табл. 1. Опис даних колонок.*

Тобто, кожна колонка містить певний параметр. Наприклад, pelvic incidence у перекладі – охоплення тазу, pelvic\_tilt – нахил тазу, етс. [3]

## Розділ 2. Етапи виконання завдання

Повний код програми можна побачити у додатку Б. Під час виконання були виділені наступні етапи:

### **Створення багатошарового персептрону.**

За допомогою бібліотеки sklearn був створений MLP, що класифікує результати відносно вимог, проте його точність (з більшістю параметрів за замовчуванням) коливалась від 70% до 80%, тому для підняття його ефективності потрібно розробити генетичний алгоритм, що сформував би оптимальний багатошаровий персептрон.

### **Реалізація генетичного алгоритму**

Був реалізований генетичний алгоритм для формування оптимального персептрону. У ході розробки були прийняти наступні рішення:

1. **Кількість поколінь, та популяція кожного покоління.**

Неможливо точно сказати оптимальну кількість цих параметрів, і їх значення, зазвичай, підбираються практичним шляхом. Але існують рекомендації для перших кроків дослідження.

**Розмір популяції:**

Менший розмір популяції забезпечує швидшу конвергенцію (стан, коли кожна особина в популяції однакова, або майже однакова, і еволюція припинилась), але тоді алгоритм може потрапити до *локального* оптимуму, що не є гарним, але не найкращим варіантом. Зворотне стосується великої чисельності популяції. [4]

Таким чином для початку рекомендують обирати або число що дорівнює:

X \* D

X – будь яке число від 5 до 10

D – кількість вимірів, у нашому випадку - кількість вхідних параметрів налаштувань.[5],

**Або**

(1.5 або 2) \* G

X – будь яке число від 1.5 до 2

G – кількість генів (можливих опцій налаштувань) [6][7].

У нашому випадку за першим засобом було отримано: 6 \* 10 = 60, а за другим: 2 \* 26 = 52. Був обраний перший варіант для забезпечення різноманітності поколінь – ***60***.

**Кількість поколінь:**

Як і у випадку з популяцією, кількість поколінь зазвичай підбирається практичним шляхом, проте рекомендується обирати число збалансоване з кількістю поколінь, наприклад 60\50, 50\60, щоб як можна меньша кількість останніх поколінь були у стані конвергенції, та не виконувалась «пуста» робота. Для даної задачі була обрана кількість поколінь - ***50*** [8]

1. **Шанс мутації**

Шанс мутації також залежить від задачі та її стадії. Зазвичай, під час перших запусків та початкового пошуку її ставлять досить високою, поступово знижуючи [9]

Крім того, була введено авторське поняття «радикальна мутація». Вона використовується для того, щоб у випадку, якщо алгоритм потрапить до *локального* оптимуму, у нього була можливість вийти зі стану конвергенції, та знайти нові, більш оптимальні налаштування. Ця мутація, на відміну від звичайної, сильно змінює певний параметр, та може кардинально змінити роботи MLP. Це рішення дало приріст ефективності для розроблюваного генетичного алгоритму.

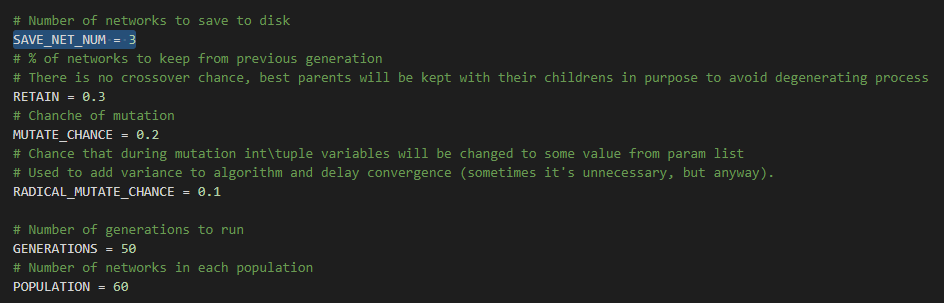
Для задачі що була поставлена у РГР був обраний шанс мутації ***0.2***, та шанс того, що ця мутація стане радикальною – ***0.1*** (тобто загальний шанс радикальної мутації = 0.2 \* 0.1)

## Розділ 3. Остаточний результат

В результаті виконання програми було збережено 3 найкращі MLP (параметер що відповідає за кількість збереженних MLP можна налаштувати разом з усіма іншими, див. рис. 3).

Ці MLP мають високу точність класифікації, та найбільш оптимізовані для заданої задачі. (рис. 5).

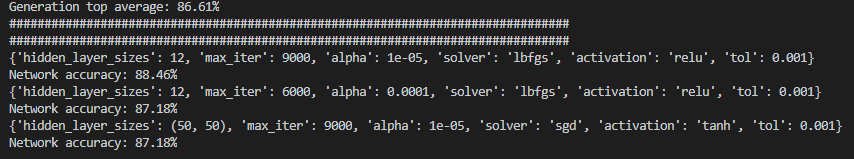
У випадку їх необхідності, модель можна бути прочитати з диску, та використовувати для класифікації (рис. 6).



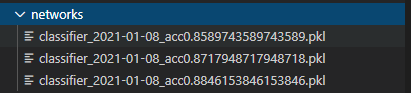
*Рис. 3. Можливі налаштування програми*

**

*Рис. 4. Процесс роботи програми*



*Рис. 5. Оптимальні налаштування MLP що знайдені за допомогою генетичного алгоритму*

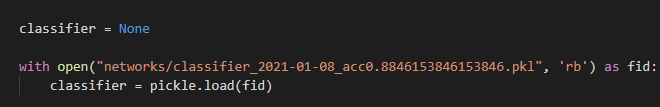
**

*Рис. 6. Збереженні MLP*

Прочитаємо записаний MLP, да дамо йому 2 записи. Перший – аномальний, а другий – нормальный (рис. 7, рис. 8, рис. 9)



*Рис. 7. Тестові дані*



*Рис. 8. Завантаження класифікатора*



*Рис. 9. Результат роботи завантаженого класифікатора*

Як можна побачити за рис. 9, класифікатор відпрацював вірно. Повний код тесту можна побачити у додатку Б.

# ВИСНОВКИ

В розрахунково-графічній роботі був створений класифікатор у вигляді багатошарового персептрону, сформованого на основі методу пошуку у вигляді генетичного алгоритму

Під час виконання завдання була використана мова програмування Python (з використанням ООП), та також використані деякі бібліотеки для створення MLP, як sklearn, що дозволило спростити та пришвидшити розробку і реалізувати завдання. Весь код було задокументовано для простишого його сприйняття.

В результаті роботи було отримано класифікатор у вигляді багатошарового персептрону, що був сформований завдяки генетичного алгоритму, що може за даними стану хребта (та тіла в загалому) пацієнта виявити, чи є аномалії у його показниках.

Так як алгоритм виконується досить довго (20-25 хв при популяції в 60 мереж та 50 поколіннях на машині з процесором: Intel® Core™ i5-4590, та 16gb RAM), для покращення програми у майбутьному можна додати параллельні обчислення за допомогою модуля multiprocessing. Крім того, ще можна продумати і покращити механізм мутацій, що дозволить формувати кращі для MLP.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Lower Back Pain Symptoms Dataset [Електронний ресурс] – Режим доступу до ресурсу: https://www.kaggle.com/sammy123/lower-back-pain-symptoms-dataset

2. What is causing this pain in my back? [Електронний ресурс] – Режим доступу до ресурсу: https://www.medicalnewstoday.com/articles/172943#diagnosis

3. An Exploratory Data Analysis on Lower Back Pain [Електронний ресурс] – Режим доступу до ресурсу: https://towardsdatascience.com/an-exploratory-data-analysis-on-lower-back-pain-6283d0b0123

4. An Investigation on Genetic Algorithm Parameters [Електронний ресурс] – Режим доступу до ресурсу: http://sarmady.com/siamak/papers/genetic-algorithm.pdf

5. What is the optimal/recommended population size for differential evolution? [Електронний ресурс] – Режим доступу до ресурсу: https://www.researchgate.net/post/What\_is\_the\_optimal\_recommended\_population\_size\_for\_differential\_evolution2

6. Vose, M.: Modeling simple genetic algorithms. Evol. Comput. 3(4), 453–472 (1996)

7. How to calculate the Crossover, Mutation rate and population size for Genetic algorithm? [Електронний ресурс] – Режим доступу до ресурсу: https://www.researchgate.net/post/How-to-calculate-the-Crossover-Mutation-rate-and-population-size-for-Genetic-algorithm

8. What is the optimal/recommended population size for differential evolution? [Електронний ресурс] – Режим доступу до ресурсу: https://www.researchgate.net/post/What\_is\_the\_optimal\_recommended\_population\_size\_for\_differential\_evolution2

9. Why is the mutation rate in genetic algorithms very small? [Електронний ресурс] – Режим доступу до ресурсу: https://www.researchgate.net/post/Why-is-the-mutation-rate-in-genetic-algorithms-very-small

# ДОДАТОК А

Через занадто великий розмір даних, у додатку буде приведена тільки їх частина.

Всі дані можна знайти за посиланням: https://www.kaggle.com/sammy123/lower-back-pain-symptoms-dataset

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Col1 | Col2 | Col3 | Col4 | Col5 | Col6 | Col7 | Col8 | Col9 | Col10 | Col11 | Col12 | Class\_att |
| 63.02782 | 22.55259 | 39.60912 | 40.47523 | 98.67292 | -0.2544 | 0.744503 | 12.5661 | 14.5386 | 15.30468 | -28.6585 | 43.5123 | Abnormal |
| 39.05695 | 10.06099 | 25.01538 | 28.99596 | 114.4054 | 4.564259 | 0.415186 | 12.8874 | 17.5323 | 16.78486 | -25.5306 | 16.1102 | Abnormal |
| 68.83202 | 22.21848 | 50.09219 | 46.61354 | 105.9851 | -3.53032 | 0.474889 | 26.8343 | 17.4861 | 16.65897 | -29.0319 | 19.2221 | Abnormal |
| 69.29701 | 24.65288 | 44.31124 | 44.64413 | 101.8685 | 11.21152 | 0.369345 | 23.5603 | 12.7074 | 11.42447 | -30.4702 | 18.8329 | Abnormal |
| 49.71286 | 9.652075 | 28.31741 | 40.06078 | 108.1687 | 7.918501 | 0.54336 | 35.494 | 15.9546 | 8.87237 | -16.3784 | 24.9171 | Abnormal |
| 40.2502 | 13.92191 | 25.12495 | 26.32829 | 130.3279 | 2.230652 | 0.789993 | 29.323 | 12.0036 | 10.40462 | -1.51221 | 9.6548 | Abnormal |
| 53.43293 | 15.86434 | 37.16593 | 37.56859 | 120.5675 | 5.988551 | 0.19892 | 13.8514 | 10.7146 | 11.37832 | -20.5104 | 25.9477 | Abnormal |
| 45.36675 | 10.75561 | 29.03835 | 34.61114 | 117.2701 | -10.6759 | 0.131973 | 28.8165 | 7.7676 | 7.60961 | -25.1115 | 26.3543 | Abnormal |
| 43.79019 | 13.53375 | 42.69081 | 30.25644 | 125.0029 | 13.28902 | 0.190408 | 22.7085 | 11.4234 | 10.59188 | -20.0201 | 40.0276 | Abnormal |
| 36.68635 | 5.010884 | 41.94875 | 31.67547 | 84.24142 | 0.664437 | 0.3677 | 26.2011 | 8.738 | 14.91416 | -1.7021 | 21.432 | Abnormal |
| 49.70661 | 13.04097 | 31.3345 | 36.66564 | 108.6483 | -7.82599 | 0.68801 | 31.3502 | 16.5097 | 15.17645 | -0.50213 | 18.3437 | Abnormal |
| 31.23239 | 17.71582 | 15.5 | 13.51657 | 120.0554 | 0.499751 | 0.608343 | 21.4356 | 9.2589 | 14.76412 | -21.7246 | 36.4449 | Abnormal |
| 48.91555 | 19.96456 | 40.26379 | 28.951 | 119.3214 | 8.028895 | 0.139478 | 32.7916 | 7.2049 | 8.61882 | -1.21554 | 27.3713 | Abnormal |
| 53.57217 | 20.46083 | 33.1 | 33.11134 | 110.9667 | 7.044803 | 0.081931 | 15.058 | 12.8127 | 12.00109 | -1.73412 | 15.6205 | Abnormal |
| 57.30023 | 24.18888 | 47 | 33.11134 | 116.8066 | 5.766947 | 0.416722 | 16.5158 | 18.6222 | 8.51898 | -33.4413 | 13.2498 | Abnormal |
| 44.31891 | 12.53799 | 36.09876 | 31.78092 | 124.1158 | 5.415825 | 0.664041 | 9.5021 | 19.1756 | 7.25707 | -32.8939 | 19.5695 | Abnormal |
| 63.83498 | 20.36251 | 54.55243 | 43.47247 | 112.3095 | -0.62253 | 0.560675 | 10.769 | 16.8116 | 11.41344 | 2.676002 | 17.3859 | Abnormal |
| 31.27601 | 3.144669 | 32.563 | 28.13134 | 129.0114 | 3.62302 | 0.534481 | 31.1641 | 18.6089 | 8.4402 | 4.482424 | 24.6513 | Abnormal |
| 38.69791 | 13.44475 | 31 | 25.25316 | 123.1593 | 1.429186 | 0.306581 | 28.3015 | 17.9575 | 14.75417 | -14.2527 | 24.9361 | Abnormal |
| 41.72996 | 12.25407 | 30.12259 | 29.47589 | 116.5857 | -1.2444 | 0.468526 | 28.5598 | 12.4637 | 14.1961 | -20.3925 | 33.0265 | Abnormal |
| 43.92284 | 14.17796 | 37.83255 | 29.74488 | 134.461 | 6.451648 | 0.280446 | 12.4719 | 16.8965 | 10.32658 | -4.98667 | 22.4667 | Abnormal |
| 54.91944 | 21.06233 | 42.2 | 33.85711 | 125.2127 | 2.432561 | 0.175245 | 23.0791 | 14.2195 | 14.14196 | 3.780394 | 24.9278 | Abnormal |
| 63.07361 | 24.4138 | 54 | 38.65981 | 106.4243 | 15.7797 | 0.666388 | 11.9696 | 17.6891 | 7.63771 | -14.1836 | 44.2338 | Abnormal |
| 45.54079 | 13.0696 | 30.29832 | 32.47119 | 117.9808 | -4.98713 | 0.56745 | 23.8889 | 9.1019 | 7.70987 | -19.379 | 20.3649 | Abnormal |
| …………… |  |  |  |  |  |  |  |  |  |  |  |  |
| 46.2364 | 10.06277 | 37 | 36.17363 | 128.0636 | -5.10005 | 0.860784 | 9.5912 | 15.1769 | 16.49989 | -22.42 | 40.2061 | Normal |
| 46.42637 | 6.620795 | 48.1 | 39.80557 | 130.3501 | 2.449382 | 0.515439 | 9.1955 | 10.6369 | 15.11344 | 2.963625 | 23.0719 | Normal |
| 39.6569 | 16.20884 | 36.67486 | 23.44806 | 131.922 | -4.96898 | 0.794717 | 31.3737 | 18.3533 | 13.16102 | -6.65262 | 26.3297 | Normal |
| 45.57548 | 18.75914 | 33.77414 | 26.81635 | 116.797 | 3.13191 | 0.514212 | 24.2526 | 12.9572 | 12.40401 | -12.3631 | 31.9668 | Normal |
| 66.50718 | 20.89767 | 31.72747 | 45.60951 | 128.9029 | 1.517203 | 0.787252 | 12.8877 | 11.8978 | 9.2322 | -14.8244 | 43.8409 | Normal |
| 82.90535 | 29.89412 | 58.25054 | 53.01123 | 110.709 | 6.079338 | 0.827146 | 12.5622 | 12.3646 | 16.61754 | -15.7588 | 35.9458 | Normal |
| 50.67668 | 6.461501 | 35 | 44.21518 | 116.588 | -0.21471 | 0.021178 | 18.7846 | 8.007 | 9.74352 | -1.2286 | 14.2547 | Normal |
| 89.01488 | 26.07598 | 69.02126 | 62.93889 | 111.4811 | 6.061508 | 0.544505 | 27.0219 | 13.3731 | 11.04819 | -3.5053 | 33.4196 | Normal |
| 54.60032 | 21.48897 | 29.36022 | 33.11134 | 118.3433 | -1.47107 | 0.962907 | 30.8554 | 11.4198 | 13.82322 | -5.60645 | 18.5514 | Normal |
| 34.3823 | 2.062683 | 32.39082 | 32.31962 | 128.3002 | -3.36552 | 0.581169 | 12.0774 | 16.6255 | 7.20496 | -31.3748 | 29.5748 | Normal |
| 45.07545 | 12.30695 | 44.58318 | 32.7685 | 147.8946 | -8.94171 | 0.932922 | 32.1169 | 14.3037 | 10.64326 | -31.1988 | 11.2307 | Normal |
| 47.90357 | 13.61669 | 36 | 34.28688 | 117.4491 | -4.2454 | 0.129744 | 7.8433 | 14.7484 | 8.51707 | -15.7289 | 11.5472 | Normal |
| 53.93675 | 20.7215 | 29.22053 | 33.21525 | 114.3658 | -0.42101 | 0.047913 | 19.1986 | 18.1972 | 7.08745 | 6.013843 | 43.8693 | Normal |
| 61.4466 | 22.69497 | 46.17035 | 38.75163 | 125.6707 | -2.70788 | 0.08107 | 16.2059 | 13.5565 | 8.89572 | 3.564463 | 18.4151 | Normal |
| 45.25279 | 8.693157 | 41.58313 | 36.55963 | 118.5458 | 0.21475 | 0.159251 | 14.7334 | 16.0928 | 9.75922 | 5.767308 | 33.7192 | Normal |
| 33.84164 | 5.073991 | 36.64123 | 28.76765 | 123.9452 | -0.19925 | 0.674504 | 19.3825 | 17.6963 | 13.72929 | 1.783007 | 40.6049 | Normal |

# ДОДАТОК Б

**Лістинг програми:**

***main.py***

"""Entry point to evolving the neural network. Start here."""

from optimizer import Optimizer

from tqdm import tqdm

import pandas as pd

from pathlib import Path

import pickle

from datetime import datetime

# Number of networks to save to disk

SAVE\_NET\_NUM = 3

# % of networks to keep from previous generation

# There is no crossover chance, best parents will be kept with their childrens in purpose to avoid degenerating process

RETAIN = 0.3

# Chanche of mutation

MUTATE\_CHANCE = 0.2

# Chance that during mutation int\tuple variables will be changed to some value from param list

# Used to add variance to algorithm and delay convergence (sometimes it's unnecessary, but anyway).

RADICAL\_MUTATE\_CHANCE = 0.1

# Number of generations to run

GENERATIONS = 50

# Number of networks in each population

POPULATION = 60

def train\_networks(networks, x, y):

    """Train each network.

    Args:

        networks (list): Current population of networks

        x (DataFrame): df of all collected data params

        y (DataFrame): df of same height with normal/abnormal classification of x

    """

    # Progress bar to display work of the program.

    pbar = tqdm(total=len(networks))

    # Train each network.

    for network in networks:

        network.train(x, y)

        pbar.update(1)

    # Close progress bar.

    pbar.close()

def get\_average\_accuracy(networks):

    """Get the average accuracy for a group of networks (calculated onle by best retained nets).

    Args:

        networks (list): List of networks

    Returns:

        float: The average accuracy of a population of networks.

    """

    total\_accuracy = 0

    # Sort from best to worst, and get num of networks to save.

    sorted\_net = sorted(networks, reverse=True, key=lambda x: x.accuracy)

    to\_retain = int(len(sorted\_net) \* RETAIN)

    for network in sorted\_net[:to\_retain]:

        total\_accuracy += network.accuracy

    # Get avg accuracy.

    return total\_accuracy / to\_retain

def generate(generations, population, nn\_param\_choices, x, y):

    """Generate a network with the genetic algorithm.

    Args:

        generations (int): Number of times to evole the population

        population (int): Number of networks in each generation

        nn\_param\_choices (dict): Parameter choices for networks

        x (DataFrame): df of all collected data params

        y (DataFrame): df of same height with normal/abnormal classification of x

    Args example:

        x:

                    Col1       Col2       Col3       Col4        Col5       Col6      Col7     Col8     Col9     Col10      Col11    Col12

            0    63.027818  22.552586  39.609117  40.475232   98.672917  -0.254400  0.744503  12.5661  14.5386  15.30468 -28.658501  43.5123

            2    33.841641   5.073991  36.641233  28.767649  123.945244  -0.199249  0.674504  19.3825  17.6963  13.72929   1.783007  40.6049

            ..         ...        ...        ...        ...         ...        ...       ...      ...      ...       ...        ...      ...

        y:

            0      Abnormal

            1      Normal

                    ...

    Returns:

        networks: the list of best networks.

    """

    # I could use rollback to old generation (if new avg acc < new avg acc),

    # but since code keeps the best networks "alive"and unchanged, result cannot go down.

    # Any light acc drop during the work of the programm - result of non-deterministic learning process of MLP.

    optimizer = Optimizer(nn\_param\_choices, retain=RETAIN, mutate\_chance=MUTATE\_CHANCE, radical\_mutate\_chance=RADICAL\_MUTATE\_CHANCE)

    networks = optimizer.create\_population(population)

    # Evolve the generation.

    for i in range(generations):

        print("\*\*\*Doing generation %d of %d\*\*\*" %(i + 1, generations))

        # Train and get accuracy for networks.

        train\_networks(networks, x, y)

        # Get the average accuracy for this generation.

        average\_accuracy = get\_average\_accuracy(networks)

        # Print out the average accuracy each generation.

        print("Generation top average: %.2f%%" % (average\_accuracy \* 100))

        # Evolve, except on the last iteration.

        if i != generations - 1:

            # Do the evolution.

            networks = optimizer.evolve(networks)

        print('#'\*80)

    # Sort our final population.

    networks = sorted(networks, key=lambda x: x.accuracy, reverse=True)

    return networks

def print\_networks(networks):

    """Print a list of networks.

    Args:

        networks (list): The population of networks

    """

    print('#'\*80)

    for network in networks:

        network.print\_network()

def get\_data():

    """Prepares data from Dataset\_spine.csv"""

    df = pd.read\_csv('Dataset\_spine.csv')

    # Drop dummy column.

    df = df.drop(['Unnamed: 13'], axis=1)

    # Rename columns according to: https://towardsdatascience.com/an-exploratory-data-analysis-on-lower-back-pain-6283d0b0123.

    df.rename(columns={

            "Col1" : "pelvic\_incidence",

            "Col2" : "pelvic\_tilt",

            "Col3" : "lumbar\_lordosis\_angle",

            "Col4" : "sacral\_slope",

            "Col5" : "pelvic\_radius",

            "Col6" : "degree\_spondylolisthesis",

            "Col7" : "pelvic\_slope",

            "Col8" : "Direct\_tilt",

            "Col9" : "thoracic\_slope",

            "Col10" : "cervical\_tilt",

            "Col11" : "sacrum\_angle",

            "Col12" : "scoliosis\_slope",

        }

    )

    # Generate set of data and it's classification set for MLP.

    y = df['Class\_att']

    x = df.drop(['Class\_att'], axis=1)

    return x, y

def save\_networks(networks, dir\_path = "./networks"):

    """Saves best networks to the disk."""

    Path(dir\_path).mkdir(parents=True, exist\_ok=True)

    for network in networks:

        file\_name = f"{dir\_path}/classifier\_{datetime.today().strftime('%Y-%m-%d')}\_acc{network.accuracy}.pkl"

        with open(file\_name, 'wb') as fid:

            pickle.dump(network, fid)

def main():

    """Evolve a network."""

    if POPULATION < SAVE\_NET\_NUM:

        raise Exception("population number must be >= ", SAVE\_NET\_NUM)

    nn\_param\_choices = {

        "hidden\_layer\_sizes": [(12), (12, 12), (12,12,12),

        (50), (50, 50), (50,50,50),

        (100), (100, 100), (100, 100, 100)],

        "max\_iter": [6000, 7000, 8000, 9000],

        "alpha": [0.001, 0.0001, 0.00001],

        "solver": ['lbfgs', 'sgd', 'adam'], # The solver for weight optimization.

        "activation" : ['identity', 'logistic', 'tanh', 'relu'],

        "tol" : [1e-3, 1e-4, 1e-5, 1e-6],

    }

    print("\*\*\*Evolving %d generations with population %d\*\*\*" %

                 (GENERATIONS, POPULATION))

    x, y = get\_data()

    networks = generate(GENERATIONS, POPULATION, nn\_param\_choices, x, y)

    # Print out the top 5 networks.

    print\_networks(networks[:SAVE\_NET\_NUM])

    save\_networks(networks[:SAVE\_NET\_NUM])

if \_\_name\_\_ == '\_\_main\_\_':

    main()

***network.py***

"""Class that represents the network to be evolved."""

import random

import logging

from sklearn.neural\_network import MLPClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score

from sklearn.metrics import confusion\_matrix

class Network():

    """Represent a network and let us operate on it.

    Currently only works for an MLP.

    """

    def \_\_init\_\_(self, nn\_param\_choices=None):

        """Initialize our network."""

        self.accuracy = 0.

        self.nn\_param\_choices = nn\_param\_choices

        self.network = {}  # (dic): represents MLP network parameters.

        self.clf = None

    def create\_random(self):

        """Create a random network."""

        for key in self.nn\_param\_choices:

            self.network[key] = random.choice(self.nn\_param\_choices[key])

    def create\_set(self, network):

        """Set network properties.

        Args:

            network (dict): The network parameters

        """

        self.network = network

    def train(self, x, y):

        """Train the network and record the accuracy.

        Args:

            x (DataFrame): df of all collected data params

            y (DataFrame): df of same height with normal/abnormal classification of x

        """

        # If model already trained - skip it.

        if self.accuracy != 0.:

            return

        # Create MLP classifier with given params.

        self.clf = MLPClassifier(\*\*self.network)

        # Get datasets for training/testing, then train and test model.

        x\_train, x\_test, y\_train, y\_test = train\_test\_split(x,y, test\_size= 0.25, random\_state=27)

        self.clf.fit(x\_train, y\_train)

        y\_pred = self.clf.predict(x\_test)

        # set curr model accuracy

        self.accuracy = accuracy\_score(y\_test, y\_pred)

    def print\_network(self):

        """Print out a network."""

        print(self.network)

        print("Network accuracy: %.2f%%" % (self.accuracy \* 100))

***optimizer.py***

"""

Class that holds a genetic algorithm for evolving a network.

"""

from functools import reduce

from operator import add

import random

from network import Network

class Optimizer():

    """Class that implements genetic algorithm for MLP optimization."""

    def \_\_init\_\_(self, nn\_param\_choices, retain=0.4,

                 random\_select=0.1, mutate\_chance=0.25, radical\_mutate\_chance=0.1):

        """Create an optimizer.

        Args:

            nn\_param\_choices (dict): Possible network paremters

            retain (float): Percentage of population to retain after

                each generation

            random\_select (float): Probability of a rejected network

                remaining in the population

            mutate\_chance (float): Probability a network will be

                randomly mutated

        """

        self.mutate\_chance = mutate\_chance

        self.radical\_mutate\_chance = radical\_mutate\_chance

        self.random\_select = random\_select

        self.retain = retain

        self.nn\_param\_choices = nn\_param\_choices

    def create\_population(self, count):

        """Create a population of random networks.

        Args:

            count (int): Number of networks to generate, aka the

                size of the population

        Returns:

            (list): Population of network objects

        """

        pop = []

        for \_ in range(0, count):

            # Create a random network.

            network = Network(self.nn\_param\_choices)

            network.create\_random()

            # Add the network to our population.

            pop.append(network)

        return pop

    @staticmethod

    def fitness(network):

        """Return the accuracy, which is our fitness function."""

        return network.accuracy

    def grade(self, pop):

        """Find average fitness for a population.

        Args:

            pop (list): The population of networks

        Returns:

            (float): The average accuracy of the population

        """

        summed = reduce(add, (self.fitness(network) for network in pop))

        return summed / float((len(pop)))

    def breed(self, mother, father):

        """Make two children as parts of their parents.

        Args:

            mother (dict): Network parameters

            father (dict): Network parameters

        Returns:

            (list): Two network objects

        """

        children = []

        for \_ in range(2):

            child = {}

            # Loop through the parameters and pick params for the kid.

            for param in self.nn\_param\_choices:

                child[param] = random.choice(

                    [mother.network[param], father.network[param]]

                )

            # Now create a network object.

            network = Network(self.nn\_param\_choices)

            network.create\_set(child)

            # Randomly mutate some of the children.

            if self.mutate\_chance > random.random():

                network = self.mutate(network)

            children.append(network)

        return children

    def int\_mutate(self, number, mutate\_percent=10):

        """Mutates integer number up to mutate\_percent from its initial value,

        as example, 10 could be mutated to a value from range [9:11].

        Args:

            number (int): Number to mutate.

            father (dict): Percent of the mutation.

        Returns:

            (int): Mutated number

        """

        percent = number \* (mutate\_percent/100)

        return random.randint(int(number - percent), int(number + percent))

    def float\_mutate(self, number, mutate\_percent=15):

        """Mutates float number up to mutate\_percent from its initial value.

        Args:

            number (float): Number to mutate.

            father (dict): Percent of the mutation.

        Returns:

            (float): Mutated number

        """

        percent = number \* (mutate\_percent/100)

        return random.uniform(number - percent, number + percent)

    def random\_layers(self, tuple\_layers):

        """Mutates tuple that represents number of neurons in every layer of MLP.

        Args:

            tuple\_layers (tuple): tuple that represents number of neurons in every layer of MLP

        Returns:

            (tuple): Mutated tuple

        """

        mutated = []

        for layer\_num in tuple\_layers:

            mutated.append(self.int\_mutate(layer\_num))

        return tuple(mutated)

    def mutate(self, network):

        """Randomly mutate one part of the network.

        Args:

            network (dict): The network parameters to mutate

        Returns:

            (Network): A randomly mutated network object

        """

        # Choose a random key.

        key = random.choice(list(self.nn\_param\_choices.keys()))

        val = network.network[key]

        # Mutate one of the params.

        if isinstance(val, tuple):

            # with a little chance - change the number of layers

            if random.random() > self.radical\_mutate\_chance:

                network.network[key] = self.random\_layers(val)

            else:

                network.network[key] = random.choice(self.nn\_param\_choices[key])

        if isinstance(val, int):

            # Chance of radical mutation

            if random.random() > self.radical\_mutate\_chance:

                network.network[key] = self.int\_mutate(val)

            else:

                network.network[key] = random.choice(self.nn\_param\_choices[key])

        if isinstance(val, float):

            if random.random() > self.radical\_mutate\_chance:

                network.network[key] = self.float\_mutate(val)

            else:

                network.network[key] = random.choice(self.nn\_param\_choices[key])

        if isinstance(val, str):

            network.network[key] = random.choice(self.nn\_param\_choices[key])

        return network

    def evolve(self, pop):

        """Evolve a population of networks.

        Args:

            pop (list): A list of network parameters

        Returns:

            (list): The evolved population of networks

        """

        # Get scores for each network.

        graded = [(self.fitness(network), network) for network in pop]

        # Sort on the scores.

        graded = [x[1] for x in sorted(graded, key=lambda x: x[0], reverse=True)]

        # Get the number we want to keep for the next gen.

        retain\_length = int(len(graded)\*self.retain)

        # The parents are every network we want to keep.

        parents = graded[:retain\_length]

        # For those we aren't keeping, randomly keep some anyway.

        for individual in graded[retain\_length:]:

            if self.random\_select > random.random():

                parents.append(individual)

        # Now find out how many spots we have left to fill.

        parents\_length = len(parents)

        desired\_length = len(pop) - parents\_length

        children = []

        # Add children, which are bred from two remaining networks.

        while len(children) < desired\_length:

            self.born\_childrens(parents, children, desired\_length)

        parents.extend(children)

        return parents

    def born\_childrens(self, parents, children, desired\_length):

        """Create childrens from given networks

        Args:

            parents (list): networks from wich childrens will be created

            children (list): list of childrens to extend

            desired\_length (int): number of childrens to born

        """

        # Get a random mom and dad.

        male = random.randint(0, len(parents)-1)

        female = random.randint(0, len(parents)-1)

        # Assuming they aren't the same network...

        if male == female:

            return

        male = parents[male]

        female = parents[female]

        # Breed them.

        babies = self.breed(male, female)

        # Add the children one at a time.

        for baby in babies:

            # Don't grow larger than desired length.

            if len(children) >= desired\_length:

                return

            children.append(baby)

***test.py***

import pandas as pd

import pickle

from network import Network

classifier = None

with open("networks/classifier\_2021-01-08\_acc0.8846153846153846.pkl", 'rb') as fid:

    classifier = pickle.load(fid)

df = pd.read\_csv('Dataset\_spine\_test.csv')

# Drop dummy column.

df = df.drop(['Unnamed: 13'], axis=1)

# Rename columns according to: https://towardsdatascience.com/an-exploratory-data-analysis-on-lower-back-pain-6283d0b0123.

df.rename(columns={

        "Col1" : "pelvic\_incidence",

        "Col2" : "pelvic\_tilt",

        "Col3" : "lumbar\_lordosis\_angle",

        "Col4" : "sacral\_slope",

        "Col5" : "pelvic\_radius",

        "Col6" : "degree\_spondylolisthesis",

        "Col7" : "pelvic\_slope",

        "Col8" : "Direct\_tilt",

        "Col9" : "thoracic\_slope",

        "Col10" : "cervical\_tilt",

        "Col11" : "sacrum\_angle",

        "Col12" : "scoliosis\_slope",

    }

)

x\_test = df.drop(['Class\_att'], axis=1)

print(x\_test)

y\_pred = classifier.clf.predict(x\_test)

print(y\_pred)