НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

Кафедра автоматизованих систем обробки інформації та управління

(повна назва кафедри, циклової комісії)

КУРСОВА РОБОТА

з «Основи машинного навчання-1»
(назва дисципліни)

Студента V курсу ІП-92мп групи
напряму підготовки 8.050103 «Програмна інженерія»
спеціальності «Програмне забезпечення систем»
Ващенка Ю. О.
(прізвище та ініціали)
Керівник Сарнацький В. В.
(прізвище та ініціали)
Асистент кафедри АСОІУ
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Національна оцінка _____
Кількість балів: _____ Оцінка: ECTS _____

Члени комісії

_____ _____
(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

_____ _____
(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

Київ - 2019 рік

АНОТАЦІЯ

Пояснювальна записка до курсової роботи: 18 сторінок, 4 рисунка, 1 таблиця, 5 посилань.

Об’єкт дослідження: алгоритми класифікації.

Мета роботи: створення та навчання різних алгоритмів машинного навчання на наборі даних «Predicting a Pulsar Star».

В даній роботі розглянуті алгоритми класифікації на наборі даних «Predicting a Pulsar Star». Продемонстровано ефективність роботи алгоритмів методу опорних векторів, пониження розмірності довільним проекціюванням, ансамлювання алгоритмом AdaBoost та їх тонке налаштування за допомогою довільного пошуку оптимальних гіперпараметрів.

Ключові слова: МАШИННЕ НАВЧАННЯ, КЛАСИФІКАЦІЯ, МЕТОД ОПОРНИХ ВЕКТОРІВ, ДОВІЛЬНЕ ПРОЕКЦІЮВАННЯ, ADABOOST, ДОВІЛЬНИЙ ПОШУК ГІПЕРПАРАМЕТРІВ.

ЗМІСТ

1 ПОСТАНОВКА ЗАДАЧІ

Для успішного виконання курсової роботи необхідно:

1. Завантажити набір даних «Predicring a Pulsar Star».
2. Провести класифікацію набору даних «Predicring a Pulsar Star» алгоритмами:
    a. Метод опорних векторів;
    b. Метод опорних векторів, використовуючи довільне проекціювання для передобробки даних;
    c. AdaBoost на основі методу опорних векторів;
    d. AdaBoost на основі методу опорних векторів, використовуючи довільне проекціювання для передобробки даних.
3. Для кожного алгоритму підібрати оптимальні гіперпараметри довільним пошуком.
4. Проаналізувати результати.

2 ТЕОРЕТИЧНІ ВІДОМОСТІ

2.1 Сучасний стан машинного навчання

Нинішній успіх ML і загальне визнання стали можливі завдяки трьом обставинам:

1) Дедалі більше в геометричній прогресії кількість даних. Воно викликає потребу в аналізі даних і є необхідною умовою для впровадження систем ML. Одночасно це кількість даних відкриває можливість для навчання, оскільки породжує велику кількість зразків (прецедентів), і це достатня умова.

2) Сформувалася необхідна процесорна база. Відомо, що рішення задач ML розпадається на дві фази. На першій виконується навчання штучної нейронної мережі (тренування). Протягом цього етапу потрібно паралельно обробити велику кількість зразків. На даний момент для цієї мети немає альтернативи графічним процесорам GPU, в переважній більшості випадків використовують GPU Nvidia. Для роботи навченої нейронної мережі можуть бути використані звичайні високопродуктивні процесори CPU. Цей розподіл функцій між типами процесорів незабаром може зазнати суттєвих змін. По-перше, вже в 2017 році Intel випустила на ринок спеціалізований процесор Nervana, який буде на порядку продуктивніше, ніж GPU. По-друге, з'являються нові типи програмованих матриць FPGA і великих спеціалізованих схем ASIC, і спеціалізований процесор Google TensorFlow Processing Unit (TPU).

3) Створення бібліотек для програмного забезпечення ML. Станом на 2019 рік їх налічується більше 50. Ось тільки деякі, найбільш відомі: TensorFlow, Theano, Keras, Lasagne, Caffe, DSSTNE, Wolfram Mathematica. Список можна продовжити. Практично всі вони підтримують прикладний інтерфейс OpenMP, мови Pyton, Java і C ++ і платформу CUDA.

Майбутня сфера застосування ML, без жодного перебільшення, неозора. В контексті Четвертої промислової революції найбільш значуща роль ML полягає в розширенні потенціалу області Business Intelligence (BI), назва якої умовно перекладається як «бізнес-аналітика».

2.2 Логістична регресія

Логістична регресія або логит-регресія (англ. Logit model) - це статистична модель, використовувана для прогнозування ймовірності виникнення деякої події шляхом підгонки даних до логістичної кривої.

Логістична регресія застосовується для прогнозування ймовірності виникнення деякої події за значеннями безлічі ознак. Для цього вводиться так звана залежна змінна y, приймаюча лише одне з двох значень - як правило, це числа 0 (подія не відбулося) і 1 (подія відбулася), і безліч незалежних змінних (також званих ознаками, предикторами або регресорів) - речових $x_1, x_2, \ldots, x_n$, на основі значень яких потрібно обчислити вірогідність прийняття того чи іншого значення залежної змінної. Як і в разі лінійної регресії, для простоти запису вводиться фіктивний ознака $x_0 = 1$. Якщо порівнювати із звичайною регресійною моделлю, то метод логістичної регресії не виконує прогноз для значень числової змінної виходячи з вибірки вихідних значень незалежної змінної. Замість цього, значеннями для функції є ймовірність того, що дане вихідне значення належить до одного із класів. Аби спростити задачу припустимо, що у нас є тільки два класу і ймовірність, яку ми будемо

визначати, вірогідності того, що певне значення належить класу "+". Таким чином, результат логістичної регресії завжди буде знаходитись в інтервалі [0, 1], як і ймовірність.

Головна суть логістичної регресії полягає в тому, що простір вихідних значень незалежної змінної може бути розділений лінійною площиною (тобто, прямою) на два класи. Якщо точніше, то під лінійною площиною при двох параметрах (два виміри) мають на увазі пряму лінію, що розділяє класи (без вигинів). У випадку трьох вимірів — це площини, і так далі. Ця межа визначається в залежності від наявних вихідних даних та навчального алгоритму. Щоб все працювало, точки вихідних даних повинні розділятися лінійною площиною на дві вищезазначених області. Якщо точки вихідних даних задовольняють цій вимозі, то їх можна назвати лінійно роздільними. Візуально це зображено на рисунку 2.1.

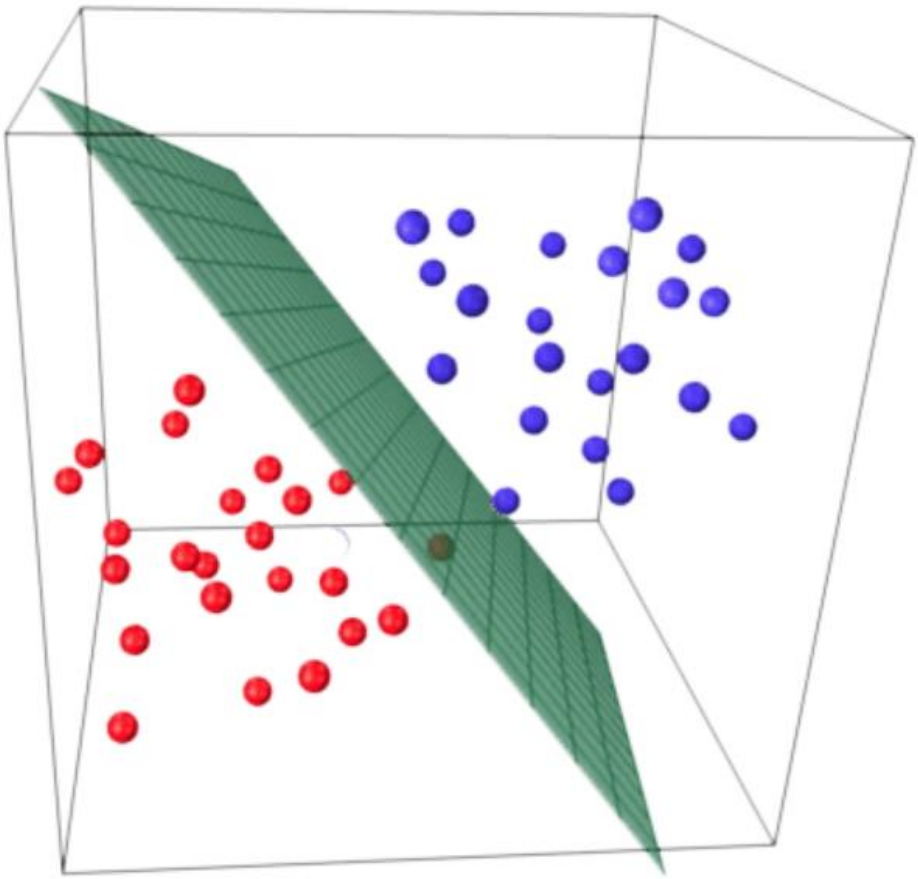

Рисунок 2.1 – Візуалізація класифікатора

Зазначена на рисунку площина називається лінійним дискримінантом. Вона є лінійною з точки зору своєї функції і моделі, а також проводить поділ, якщо точніше то дискримінацію точок на різні класи. Якщо неможливо провести лінійний розподіл точок у вихідному просторі або вони взагалі лінійно нероздільні, то варто спробувати перетворити вектори ознак в простір більшої розмірності, додавши додаткові ефекти взаємодії, члени більш високого ступеня та інше. Такий метод називають ”трюк з ядром” (Kernel trick). Використання лінійного алгоритму в такому випадку дає певні переваги для навчання нелінійної функції, оскільки межа стає нелінійною при поверненні у вихідний простір[1].

2.3 Метод k-найближчих сусідів

Метод k-найближчих сусідів — це алгоритм машинного навчання для автоматичної класифікації об'єктів, заснований на метричних показниках. Основна суть методу найближчих сусідів полягає в тому, що об’єкту присвоюється клас, який є найбільш поширеним серед сусідніх елементів даного елемента. Сусіди беруться, виходячи з множини об'єктів, класи яких уже відомі, і за допомогою гіперпараметру k. Цей гіперпараметр вираховує, який клас є найчисленнішим серед сусідів. Кожен об'єкт має кінцеву кількість атрибутів. Це алгоритм навчання з учителем, тому для роботи алгоритму потрібна розмічена вибірка даних.

Метод k-найближчих сусідів (англ. k-nearest neighbor method) — це метод класифікації даних, що є непараметричним. Для вирішення задачі класифікації об'єктів у рамках простору властивостей використовуються різні відстані 17 . (евклідові, відстань Мінковського), пораховані до усіх інших об'єктів. Вибираються об'єкти, до яких відстань найменша, і вони виділяються в окремий клас. Приклад роботи класифікатора наведено на рисунку 2.2.

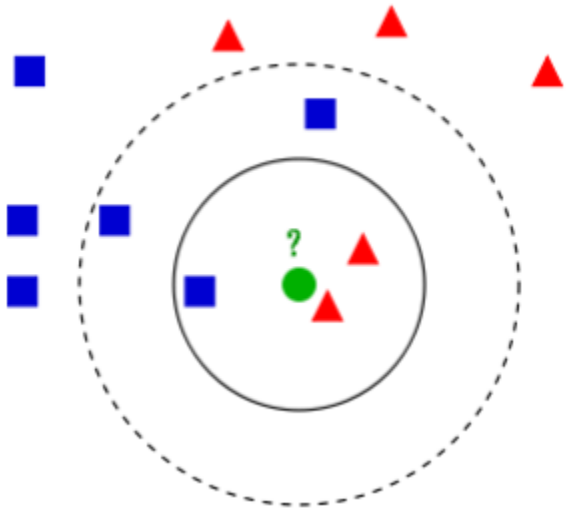

Рисунок 2.2 — Візуалізація класифікатора

Вибірка складається з двох класів: синій квадрат (клас 1) та червоний трикутник (клас 2). Потрібно класифікувати зелене коло до одного з цих класів. Якщо гіперпараметр k = 3, то зелене коло буде віднесено до другого класу, тому що всередині меншого кола два трикутника і тільки один квадрат. Якщо гіперпараметр k = 5, то коло буде віднесено до другого класу, бо всередині більшого кола три квадрата проти двох трикутників[2].

Алгоритм може бути застосований до вибірок з великою кількістю атрибутів (багатовимірним). Для цього перед застосуванням потрібно визначити функцію дистанції. Класичний варіант визначення дистанції - дистанція в евклідовому просторі.

Коли вхідні дані в алгоритм занадто великі, щоб оброблятись, і підозрюється, що вони є зайвими (наприклад, однакові вимірювання як у футах, так і в метрах), то вхідні дані будуть перетворені на зменшений набір функцій (також названий вектор функцій). Перетворення вхідних даних у набір функцій називається вилученням функції. Якщо вилучені функції ретельно вибираються, очікується, що набір функцій витягне відповідну інформацію з вхідних даних для виконання потрібної задачі, використовуючи це зменшене подання замість вводу повного розміру. Вилучення особливостей здійснюється на необроблених даних до застосування алгоритму k-NN на трансформованих даних у просторі функцій.

2.4 Метод опорних векторів (SVM)

Метод опорних векторів (англ. SVM, support vector machine) - набір схожих алгоритмів навчання з учителем, що використовуються для задач класифікації та регресійного аналізу. Належить сімейству лінійних класифікаторів і може також розглядатися як спеціальний випадок регуляризації по Тихонову. Особливою властивістю методу опорних векторів

є невпинне зменшення емпіричної помилки класифікації і збільшення відстані (margin), тому метод також відомий як метод класифікатора з максимальною відстанню. Принцип роботи наступний: Враховуючи набір навчальних прикладів, кожен з яких позначається як такий, що належить до тієї чи іншої з двох категорій, алгоритм навчання SVM створює модель, яка призначає нові приклади однієї категорії або іншої, що робить його неімовірнисним бінарним лінійним класифікатором (хоча методи наприклад, масштабування Platt існує для використання SVM в імовірнісному класифікації). SVM-модель являє собою представлення прикладів як точок в просторі, закріплені таким чином, що приклади окремих категорій ділиться явним розділенням, яке є максимально можливим. Нові дані потім вкладаються в той самий простір і передбачають, що вони належать до категорії, на підставі яких була сформована тренувальна вибірка. Крім виконання лінійної класифікації, SVM може ефективно виконувати нелінійну класифікацію, використовуючи те, що називається ”трюком з ядром” (kernel trick), що неявно відображає їхні входи у високорозмірних просторах.

Коли дані не є розміченими, навчання з учителем неможливе, і виникає необхідність у застосування спонтанного навчання, яке намагається знайти природну кластеризацію даних груп, а потім класифікувати нові дані до цих сформованих груп. Тобто, фактично, вирішується спочатку задача кластеризації одним із методів (метод к-середніх, метод ієрархічного кластерування), а вже потім безпосередньо задача класифікації. Алгоритм векторної кластеризації, створений Хавою Сігельманом та Володимиром Вапніком, застосовує статистику векторів підтримки, розроблених в алгоритмі векторних машин підтримки, для класифікації нерозмічених даних і є одним з

найбільш широко використовуваних алгоритмів кластеризації в промислових цілях. Приклад роботи класифікатора наведено на рисунку 2.3.

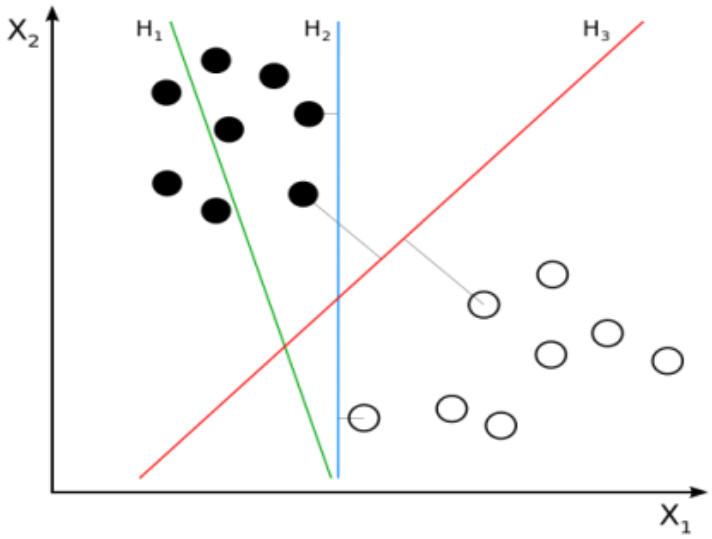

Рисунок 2.3 – Візуалізація класифікатора

Гіперплощина H1 не є роздільною. Гіперплощина H2 є роздільною, але не з максимальним розділенням. H3 є роздільною гіперплощиною із максимальним розділенням.[3]

3 МАТЕМАТИЧНІ МОДЕЛІ

3.1 Метод опорних векторів

В нас є тренувальний набір даних з n точок вигляду:

$(\vec{x}1 , y1), \dots , (\vec{x}n , yn)$

Тут yi є або 1, або -1, і кожна з цих точок вказує нам клас, до якого належить точка $\vec{xi}$. Кожна $\vec{xi}$ є p-вимірним дійсним вектором. Задача описується як задача класифікації, де нам потрібно вірно відкласифікувати вхідні дані. Для цього потрібно знайти максимальну розділову гіперплощину, яка відділяє групу точок $\vec{xi}$, де yi =1 , від групи точок, для яких yi = -1 , і визначається таким чином, що відстань між цією гіперплощиною та найближчою точкою $\vec{xi}$ з кожної з груп є максимальною.

Ми знаємо, що будь-яку гіперплощину можна записати як множину точок x , які задовольняють рівності $\vec{\omega} \cdot x - b = 0$, де $\vec{\omega}$ є (не обов'язково нормалізованим) вектором нормалі до цієї гіперплощини. Параметр $b/\|\vec{\omega}\|$ визначає зсув гіперплощини від початку координат вздовж вектора нормалі $\vec{\omega}$.

Тренувальні дані можуть бути лінійно роздільними. Тоді ми можемо обрати дві паралельні гіперплощини, які розділяють два класи даних таким чином, що відстань між ними є якомога більшою. Утвориться область, обмежена гіперплощинами. Її ми називаємо роздільною (англ. margin), а максимально розділова гіперплощина є гіперплощиною, яка лежить посередині між двома роздільними площинами кожного з класів. Ці гіперплощини може бути описані рівняннями:

$\vec{\omega} \cdot x - b = 1$ та $\vec{\omega} \cdot x - b = -1$

4 РЕАЛІЗАЦІЯ

Дана курсова робота складається з наступних компонентів:

- Клас SVM, який інкапсулює результати роботи методу опорних векторів;
- Функція iteration, яка один раз виконує навчання методу класифікації на заданих тренувальній та тестовій вибірках;
- Функція accuracy, яка повертає відсоток правильних передбачень у порівнянні з істиними значеннями;
- Функція predict, яка виконує передбачення;
- Функція grad, яка рахує градієнт;
- Функція trainSVM, яка тренує один простий метод опорних векторів для бустінгу;
- Функція adaboost, яка реалізую метод бустінгу AdaBoost;
- Функція test_hyper_parameters, яка повертає результат роботи алгоритму з заданими гіперпараметрами;
- Функція random_search, яка реалізує алгоритм довільного пошуку гіперпараметрів.

5 АНАЛІЗ РЕЗУЛЬТАТІВ

Результати виконання даної роботи наступні:

- Метод опорних векторів досягнув точності в 96.71% на наборі даних «Predicting a Pulsar Star», що краще за алгоритм випадкового вгадування (50%);
- Метод опорних векторів з препроцесінгом вхідних даних довільним проекціюванням на п’ятивимірний простів показує дещо гірші результати (94.12%), що викликано малою кількістю вхідних даних та їх високим впливом;
- Алгоритми AdaBoost на основі методу опорних векторів пришвидшив навчання, але на точність вплинув несуттєво (96.84%);
- Довільний пошук гіперпараметрів виявився не оптимальним, оскільки потребує значний час на перебір до найкращого результату, але його можливо запустити паралельно.

Результати роботи всіх моделей наведені на рисунку 3.1 та в таблиці 5.1.

Таблиця 5.1 – Зведена таблиця результатів алгоритмів

Алгоритм	Одинична модель	З AdaBoost
Метод опорних векторів	96.71%	96.84%
Метод опорних векторів з довільним проекціюванням	94.12%	94.54%

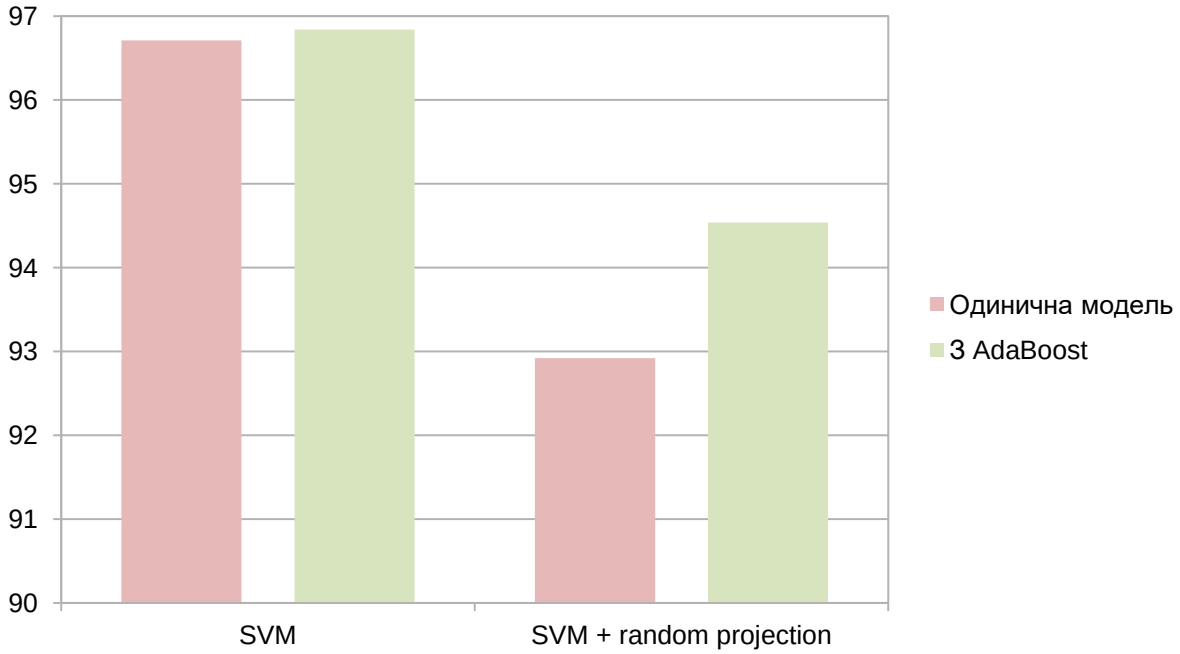

Рисунок 5.1 - Точність роботи алгоритмів

ВИСНОВКИ

В даній курсові роботі були реалізовані різні підходи для класифікації на наборі даних «Predicting a Pulsar Star». Було продемонстровано роботу методу опорних векторів, довільного проекціювання, бустингу AdaBoost та довільного пошуку оптимальних значення гіперпараметрів.

ПОСИЛАННЯ

1. Логістична регресія. Вікіпедія [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Логістична_регресія.
2. Метод k-найближчих сусідів. Вікіпедія [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Метод_k-найближчих_сусідів.
3. Метод опорних векторів. Вікіпедія [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Метод-_опорних_векторів.
4. AdaBoost. Вікіпедія [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/AdaBoost
5. Алгоритм AdaBoost. Data science. [Електронний ресурс] – Режим доступу до ресурсу: http://datascientist.one/adaboost-algorithm/

ДОДАТОК А ВИХІДНИЙ КОД

```
import numpy as np
import pandas as pd
from keras.utils import to_categorical
import common

range10000 = range(10000)

def accuracy(y_pred, y_true):
    return (y_pred == y_true).mean()

def h(w, b, x):
    return x @ w - b

def predict(w, b, x):
    return h(w, b, x).argmax(axis=1)

def H(x):
    return np.sign(x) / 2 + 1 / 2

def grad(w, b, x, y, c):
    hep = h(w, b, x)

    dw1 = -c * ((H(1 - y.T[0] * hep.T[0]) * y.T[0])[:, np.newaxis] *
x).sum(axis=0) + 2 * w.T[0]
```

```
    dw2 = -c * ((H(1 - y.T[1] * hep.T[1]) * y.T[1])[:, np.newaxis] *
x).sum(axis=0) + 2 * w.T[1]

    return (
        np.stack([dw1, dw2], axis=1),
        c * (H(1 - y * hep) * y).sum(axis=0)
    )

def iteration(x_train, y_train, x_test, y_test):
    encoded_y_train = to_categorical(y_train)

    w = np.zeros(shape=(8, 2))
    b = np.zeros(2)

    c = 0.01
    lr = 0.0001

    for i in range(encoded_y_train.T[0].size):
        for j in range(encoded_y_train[0].size):
            if encoded_y_train[i][j] == 0:
                encoded_y_train[i][j] = -1

    for i in range10000:
        dw, db = grad(w, b, x_train, encoded_y_train, c)
        w = w - lr * dw
        b = b - lr * db
```

```
    return accuracy(predict(w, b, x_test), y_test) * 100

def execute():
    (x, y) = common.getDataSet()
    piece10 = int(x.shape[0] / 10)
    result_accuracy = np.array([])

    for k in range(10):
        print(k)
        start = int(k * piece10)
        x_test = x[start: start + piece10]
        y_test = y[start: start + piece10]

        x_train_part_1 = x[0: start]
        x_train_part_2 = x[start + piece10: x.shape[0] - 1]
        x_train = np.append(x_train_part_1, x_train_part_2, axis=0)

        y_train_part_1 = y[0: start]
        y_train_part_2 = y[start + piece10: y.shape[0] - 1]
        y_train = np.append(y_train_part_1, y_train_part_2, axis=0)

        result_accuracy = np.append(result_accuracy, iteration(x_train, y_train,
x_test, y_test))

    return result_accuracy.mean()
```

```
def output():
    print("Resulted accuracy: %.2f" % execute() + "%")

output()

import numpy as np
import pandas as pd
from keras.utils import to_categorical

def one_cycle(x_train, y_train, x_test, y_test):
    encoded_y_train = to_categorical(y_train)

    def accuracy(y_pred, y_true):
        return (y_pred == y_true).mean()

    def h(w, b, x):
        return x @ w - b

    def predict(w, b, x):
        return h(w, b, x).argmax(axis=1)

    def H(x):
        return np.sign(x) / 2 + 1 / 2

    def grad(w, b, x, y, c):
```

```
        hep = h(w, b, x)

        dw1 = -c * ((H(1 - y.T[0] * hep.T[0]) * y.T[0])[:, np.newaxis] *
x).sum(axis=0) + 2 * w.T[0]
        dw2 = -c * ((H(1 - y.T[1] * hep.T[1]) * y.T[1])[:, np.newaxis] *
x).sum(axis=0) + 2 * w.T[1]

        return (
            np.stack([dw1, dw2], axis=1),
            c * (H(1 - y * hep) * y).sum(axis=0)
        )

    w = np.zeros(shape=(5, 2))
    b = np.zeros(2)

    c = 0.01
    lr = 0.0001

    for i in range(encoded_y_train.T[0].size):
        for j in range(encoded_y_train[0].size):
            if encoded_y_train[i][j] == 0:
                encoded_y_train[i][j] = -1

    for i in range(10000):
        dw, db = grad(w, b, x_train, encoded_y_train, c)
        w = w - lr * dw
```

```
        b = b - lr * db

    return accuracy(predict(w, b, x_test), y_test) * 100


df = pd.read_csv('pulsar_stars.csv')
x = df.values

y = x[:, 8]
x = x[:, :8]

M = np.random.normal(0, 0.2, size=(8, 5))
x = x @ M

result_accuracy = np.array([])

for k in range(10):
    x_test = x[int(k * (x.shape[0] / 10)): int((k + 1) * (x.shape[0] / 10))]
    y_test = y[int(k * (y.shape[0] / 10)): int((k + 1) * (y.shape[0] / 10))]

    x_train_part_1 = x[0: int(k * (x.shape[0] / 10))]
    x_train_part_2 = x[int((k + 1) * (x.shape[0] / 10)): x.shape[0] - 1]
    x_train = np.append(x_train_part_1, x_train_part_2, axis=0)

    y_train_part_1 = y[0: int(k * (y.shape[0] / 10))]
    y_train_part_2 = y[int((k + 1) * (y.shape[0] / 10)): y.shape[0] - 1]
```

```
    y_train = np.append(y_train_part_1, y_train_part_2, axis=0)

    result_accuracy = np.append(result_accuracy, one_cycle(x_train, y_train, x_test, y_test))

print("Resulted accuracy: %.2f" % result_accuracy.mean() + "%")

import numpy as np
import pandas as pd
from keras.utils import to_categorical


def one_cycle(x_train, y_train, x_test, y_test):

    def accuracy(y_pred, y_true):
        return (y_pred == y_true).mean()

    def h(w, b, x):

        return x @ w - b

    def predict(w, b, x):
        return h(w, b, x).argmax(axis=1)

    def H(x):
        return np.sign(x) / 2 + 1/2
```

```
    def grad(w, b, x, y, c):

        hep = h(w, b, x)

        dw1 = -c * ((H(1 - y.T[0] * hep.T[0]) * y.T[0])[:, np.newaxis] *
x).sum(axis=0) + 2 * w.T[0]
        dw2 = -c * ((H(1 - y.T[1] * hep.T[1]) * y.T[1])[:, np.newaxis] *
x).sum(axis=0) + 2 * w.T[1]

        return (
            np.stack([dw1, dw2], axis=1),
            c * (H(1 - y * hep) * y).sum(axis=0)
        )

    class SVM:
        def __init__(self, w, b):
            self.w = w
            self.b = b

        def predict(self, x):
            return predict(self.w, self.b, x)

        def predictPlusMin(self, x):
            return self.predict(x) * 2 - 1
```

```
def trainSVM(x, y):
    w = np.random.normal(size=(8, 2))
    b = np.random.normal(size=(2,))

    c = 0.01
    lr = 0.0001

    encoded_y = to_categorical(y)

    encoded_y = (encoded_y * 2) - 1

    for i in range(500):
        dw, db = grad(w, b, x, encoded_y, c)
        w = w - lr * dw
        b = b - lr * db

    return SVM(w, b)

def adaboost(x, y, steps=1):

    svms = list()

    _x = x
    _y = y
    y_ada = (y * 2) - 1
```

```
    for _ in range(steps):
        svm = trainSVM(_x, _y)
        y_pred = svm.predict(_x)

        y_pred = (y_pred * 2) - 1

        error = (y_ada != y_pred).mean()

        W = np.log2((1 - error) / (error + 1e-8)) / 2

        svms.append((svm, W))

        w = (1 / x.shape[0]) * np.exp(W * -(y_pred * y_ada))
        w = w / w.sum()
        idx = np.random.choice(_x.shape[0], p=w, size=(_x.shape[0]))

        _x = _x[idx]
        _y = _y[idx]
        y_ada = y_ada[idx]

    return svms

def adaPredict(svms, x):
    y_preds = [svm.predictPlusMin(x) * W for svm, W in svms]
    y_preds = np.stack(y_preds, axis=1).sum(axis=1)
    y_preds = (np.sign(y_preds) + 1) / 2
```

```
        return y_preds

    svms = adaboost(x_train, y_train, steps=10)

    return accuracy(adaPredict(svms, x_test), y_test) * 100


df = pd.read_csv('pulsar_stars.csv')
x = df.values

y = x[:, 8]
x = x[:, :8]

result_accuracy = np.array([])

for k in range(10):
    x_test = x[int(k * (x.shape[0] / 10)): int((k + 1) * (x.shape[0] / 10))]
    y_test = y[int(k * (y.shape[0] / 10)): int((k + 1) * (y.shape[0] / 10))]

    x_train_part_1 = x[0: int(k * (x.shape[0] / 10))]
    x_train_part_2 = x[int((k + 1) * (x.shape[0] / 10)): x.shape[0] - 1]
    x_train = np.append(x_train_part_1, x_train_part_2, axis=0)

    y_train_part_1 = y[0: int(k * (y.shape[0] / 10))]
    y_train_part_2 = y[int((k + 1) * (y.shape[0] / 10)): y.shape[0] - 1]
    y_train = np.append(y_train_part_1, y_train_part_2, axis=0)
```

```
    result_accuracy = np.append(result_accuracy, one_cycle(x_train, y_train, x_test, y_test))


print("Resulted accuracy: %.2f" % result_accuracy.mean() + "%")

import numpy as np
import pandas as pd
from keras.utils import to_categorical
import random

def test_hyper_parameters(x, y, c, lr):
    def one_cycle(x_train, y_train, x_test, y_test, c, lr):
        encoded_y_train = to_categorical(y_train)

        def accuracy(y_pred, y_true):
            return (y_pred == y_true).mean()

        def h(w, b, x):
            return x @ w - b

        def predict(w, b, x):
            return h(w, b, x).argmax(axis=1)

        def H(x):
```

```
        return np.sign(x) / 2 + 1 / 2

    def grad(w, b, x, y, c):
        hep = h(w, b, x)

        dw1 = -c * ((H(1 - y.T[0] * hep.T[0]) * y.T[0])[:, np.newaxis] *
x).sum(axis=0) + 2 * w.T[0]
        dw2 = -c * ((H(1 - y.T[1] * hep.T[1]) * y.T[1])[:, np.newaxis] *
x).sum(axis=0) + 2 * w.T[1]

        return (
            np.stack([dw1, dw2], axis=1),
            c * (H(1 - y * hep) * y).sum(axis=0)
        )

    w = np.zeros(shape=(8, 2))
    b = np.zeros(2)

    c = 0.01
    lr = 0.0001

    for i in range(encoded_y_train.T[0].size):
        for j in range(encoded_y_train[0].size):
            if encoded_y_train[i][j] == 0:
                encoded_y_train[i][j] = -1
```

```
        for i in range(500):
            dw, db = grad(w, b, x_train, encoded_y_train, c)
            w = w - lr * dw
            b = b - lr * db

        return accuracy(predict(w, b, x_test), y_test) * 100

    result_accuracy = np.array([])

    for k in range(10):
        x_test = x[int(k * (x.shape[0] / 10)): int((k + 1) * (x.shape[0] / 10))]
        y_test = y[int(k * (y.shape[0] / 10)): int((k + 1) * (y.shape[0] / 10))]

        x_train_part_1 = x[0: int(k * (x.shape[0] / 10))]
        x_train_part_2 = x[int((k + 1) * (x.shape[0] / 10)): x.shape[0] - 1]
        x_train = np.append(x_train_part_1, x_train_part_2, axis=0)

        y_train_part_1 = y[0: int(k * (y.shape[0] / 10))]
        y_train_part_2 = y[int((k + 1) * (y.shape[0] / 10)): y.shape[0] - 1]
        y_train = np.append(y_train_part_1, y_train_part_2, axis=0)

        result_accuracy = np.append(result_accuracy, one_cycle(x_train, y_train,
x_test, y_test, c, lr))

    return result_accuracy.mean()
```

```
def random_search(param_grid):

    best_result = 0
    best_c = 0
    best_lr = 0

    for i in range(50):
        random_params = {k: random.sample(v, 1)[0] for k, v in
param_grid.items()}

        result_accuracy = test_hyper_parameters(x, y, random_params['c'],
random_params['lr'])

        if result_accuracy > best_result:
            best_result = result_accuracy
            best_c = random_params['c']
            best_lr = random_params['lr']

    print(best_c)
    print(best_lr)



df = pd.read_csv('pulsar_stars.csv')
x = df.values
```

```
y = x[:, 8]
x = x[:, :8]

param_grid = {
   'lr': list(np.logspace(np.log10(0.00005), np.log10(0.5), base=10, num=1000)),
   'c': list(np.logspace(np.log10(0.005), np.log10(0.5), base = 10, num = 1000)),
}

random_search(param_grid)
```