به نام خدا

نام و نام خانوادگی: امیرحسین عبدیان

شماره دانشجویی: 4021661210

پرسش2 بخش آ:

کد زیر به زبان پایتون نوشته شده و از کتابخانه‌های scipy.io و numpy استفاده می‌کند. این کد داده‌های دو کلاس مختلف (کلاس سالم و کلاس دارای عیب) را از فایل‌های مربوطه بارگذاری کرده و سپس دیتاست‌هایی با تعداد نمونه و طول مشخص از این داده‌ها استخراج می‌کند.

این دو خط، کتابخانه‌های loadmat از scipy.io و numpy را وارد می‌کنند که به ترتیب برای بارگذاری داده‌ها از فایل‌های MATLAB و عملیات‌های عددی استفاده می‌شوند.

from scipy.io import loadmat

import numpy as np

این بخش، داده‌های کلاس سالم (97.mat) و کلاس دارای عیب (IR007.mat) را از فایل‌های MATLAB بارگذاری کرده و آن‌ها را به متغیرهای normal\_data و ir\_data اختصاص می‌دهد. داده‌ها در این فایل‌ها در قالب ماتریس‌های X097\_DE\_time و X105\_DE\_time ذخیره شده‌اند.

normal\_data = loadmat('/content/97.mat')['X097\_DE\_time']  # داده‌های کلاس سالم

ir\_data = loadmat('/content/IR007.mat')['X105\_DE\_time']  # داده‌های کلاس عیب

در اینجا تعداد نمونه‌های مورد نظر از هر کلاس (M = 100) و طول هر نمونه (N = 200) تعیین می‌شود.

M = 100  # تعداد نمونه‌ها از هر کلاس

N = 200  # طول هر نمونه

این خطوط بررسی می‌کنند که آیا داده‌های بارگذاری شده به اندازه کافی طولانی هستند تا تعداد نمونه‌های مورد نظر را برداشت کنند. اگر تعداد داده‌ها کافی نباشد، برنامه با یک پیام خطا متوقف می‌شود.

assert normal\_data.shape[0] >= M \* N, "Not enough data in NORMAL\_0"

assert ir\_data.shape[0] >= M \* N, "Not enough data in IR007\_x"

دو ماتریس با ابعاد (M, N) برای ذخیره داده‌های هر کلاس ایجاد می‌شوند. این ماتریس‌ها در ابتدا با صفرها پر شده‌اند.

normal\_dataset = np.zeros((M, N))

ir\_dataset = np.zeros((M, N))

در این بخش، M نمونه از داده‌های هر کلاس به صورت تصادفی انتخاب و در ماتریس‌های مربوطه ذخیره می‌شوند. برای هر نمونه، یک شاخص شروع تصادفی انتخاب شده و به اندازه N داده از آن شاخص برداشت می‌شود.

for i in range(M):

    start\_index = np.random.randint(0, normal\_data.shape[0] - N + 1)

    normal\_dataset[i, :] = normal\_data[start\_index:start\_index + N].flatten()  # نمونه‌های کلاس NORMAL\_0

    start\_index = np.random.randint(0, ir\_data.shape[0] - N + 1)

    ir\_dataset[i, :] = ir\_data[start\_index:start\_index + N].flatten()  # نمونه‌های کلاس IR007\_x

در نهایت، ابعاد ماتریس‌های ایجاد شده برای داده‌های هر کلاس چاپ می‌شوند تا کاربر مطمئن شود که نمونه‌ها به درستی استخراج شده‌اند.

print("Normal dataset shape:", normal\_dataset.shape)

print("IR dataset shape:", ir\_dataset.shape)

پرسش2 بخش ب:

این خط، کتابخانه numpy را وارد می‌کند که برای انجام عملیات‌های عددی در پایتون استفاده می‌شود.

import numpy as np

این خط، تابع calculate\_features را تعریف می‌کند که ویژگی‌های مختلفی را از داده‌ها استخراج می‌کند. اولین ویژگی که محاسبه می‌شود انحراف معیار (std\_dev) است که با استفاده از تابع np.std محاسبه می‌شود.

def calculate\_features(data):

    # انحراف معیار

    std\_dev = np.std(data, axis=1)

این خط بیشترین مقدار هر نمونه در داده‌ها را با استفاده از np.max محاسبه می‌کند و در متغیر peak ذخیره می‌کند.

    peak = np.max(data, axis=1)

این خط مقدار کجی (Skewness) را برای هر نمونه محاسبه می‌کند. کجی نشان‌دهنده‌ی نامتقارنی توزیع داده‌ها است.

    skewness = np.sum((data - np.mean(data, axis=1, keepdims=True))\*\*3, axis=1) / (std\_dev\*\*3 \* data.shape[1])

این خط کرتوزیس (Kurtosis) را برای هر نمونه محاسبه می‌کند. کرتوزیس نشان‌دهنده‌ی پهنای دم‌های توزیع داده‌ها است.

    kurtosis = np.sum((data - np.mean(data, axis=1, keepdims=True))\*\*4, axis=1) / (std\_dev\*\*4 \* data.shape[1])

این دو خط ابتدا مقدار ریشه میانگین مربعات (rms) را محاسبه می‌کنند و سپس عامل شکل (shape\_factor) را به عنوان نسبت rms به میانگین مطلق داده‌ها محاسبه می‌کنند.

rms = np.sqrt(np.mean(data\*\*2, axis=1))

    shape\_factor = rms / np.mean(np.abs(data), axis=1)

این خط عامل تأثیر (Impact Factor) را محاسبه می‌کند که نسبت بیشترین مقدار به میانگین مطلق داده‌ها است.

    impact\_factor = peak / np.mean(np.abs(data), axis=1)

این خط میانگین مطلق (Absolute Mean) داده‌ها را محاسبه می‌کند.

    absolute\_mean = np.mean(np.abs(data), axis=1)

این خط بیشترین میزان تغییرات (Peak to Peak) را محاسبه می‌کند که تفاوت بین بیشترین و کمترین مقدار داده‌ها است.

    peak\_to\_peak = np.ptp(data, axis=1)

این دو خط تمام ویژگی‌های محاسبه‌شده را در یک ماتریس ترکیب کرده و آن را به عنوان خروجی تابع برمی‌گرداند.

features = np.column\_stack((std\_dev, peak, skewness, kurtosis, shape\_factor, impact\_factor, absolute\_mean, peak\_to\_peak))

    return features

این دو خط ویژگی‌ها را برای داده‌های سالم (normal\_dataset) و داده‌های دارای عیب (ir\_dataset) با استفاده از تابع calculate\_features استخراج می‌کنند.

normal\_features = calculate\_features(normal\_dataset)

ir\_features = calculate\_features(ir\_dataset)

این خط نام‌های ویژگی‌ها را در یک لیست ذخیره می‌کند تا بتوان در چاپ خروجی‌ها از آن‌ها استفاده کرد.

feature\_names = ["Standard Deviation", "Peak", "Skewness", "Kurtosis", "Shape Factor", "Impact Factor", "Absolute Mean", "Peak to Peak"]

این بخش ویژگی‌های استخراج شده برای چند نمونه اول (حداکثر 5 نمونه) از هر دیتاست را چاپ می‌کند. ابتدا ویژگی‌های مربوط به داده‌های سالم و سپس ویژگی‌های مربوط به داده‌های دارای عیب نمایش داده می‌شوند.

print("Features for Normal Dataset:")

for i in range(min(5, normal\_features.shape[0])):  # چاپ 5 نمونه اول یا کمتر

    print(f"Sample {i+1}:")

    for j, name in enumerate(feature\_names):

        print(f"  {name}: {normal\_features[i, j]:.4f}")

print("\nFeatures for IR Dataset:")

for i in range(min(5, ir\_features.shape[0])):

    print(f"Sample {i+1}:")

    for j, name in enumerate(feature\_names):

        print(f"  {name}: {ir\_features[i, j]:.4f}")

پرسش2 بخش ج:

این خطوط، کتابخانه numpy برای عملیات‌های عددی و shuffle از sklearn.utils برای مخلوط کردن داده‌ها را وارد می‌کنند.

import numpy as np

from sklearn.utils import shuffle

در این بخش، ویژگی‌های هر دو کلاس (سالم و دارای عیب) با استفاده از np.vstack در یک ماتریس واحد ترکیب می‌شوند. همچنین برچسب‌های مربوط به این داده‌ها نیز با استفاده از np.concatenate ایجاد می‌شوند؛ برای داده‌های سالم برچسب 0 و برای داده‌های دارای عیب برچسب 1 تخصیص داده می‌شود.

features = np.vstack((normal\_features, ir\_features))

labels = np.concatenate((np.zeros(normal\_features.shape[0]), np.ones(ir\_features.shape[0])))

در این خط، داده‌ها و برچسب‌ها با استفاده از تابع shuffle مخلوط می‌شوند تا ترتیب نمونه‌ها به صورت تصادفی درآید. random\_state=42 برای اطمینان از تکرارپذیری نتایج استفاده می‌شود.

features, labels = shuffle(features, labels, random\_state=42)

در این قسمت، نسبت تقسیم داده‌ها به دو بخش آموزش و ارزیابی تعیین می‌شود. train\_ratio = 0.75 نشان می‌دهد که 75 درصد داده‌ها برای آموزش و 25 درصد برای ارزیابی استفاده خواهند شد. n\_train تعداد نمونه‌های آموزشی را مشخص می‌کند.

train\_ratio = 0.75

n\_train = int(len(features) \* train\_ratio)

این خطوط داده‌ها و برچسب‌ها را بر اساس تعداد نمونه‌های آموزشی (n\_train) به دو بخش آموزشی و ارزیابی تقسیم می‌کنند. features[:n\_train] و labels[:n\_train] به ترتیب داده‌ها و برچسب‌های بخش آموزشی و features[n\_train:] و labels[n\_train:] به ترتیب داده‌ها و برچسب‌های بخش ارزیابی هستند.

X\_train, X\_test = features[:n\_train], features[n\_train:]

y\_train, y\_test = labels[:n\_train], labels[n\_train:]

این خطوط ابعاد ماتریس‌های داده‌ها و برچسب‌های آموزشی و ارزیابی را چاپ می‌کنند تا اطمینان حاصل شود که داده‌ها به درستی تقسیم شده‌اند.

# چاپ ابعاد دیتاست‌های آموزشی و ارزیابی

print("Training data shape:", X\_train.shape)

print("Training labels shape:", y\_train.shape)

print("Test data shape:", X\_test.shape)

print("Test labels shape:", y\_test.shape)

پرسش2 بخش د:

این خط کتابخانه StandardScaler از sklearn.preprocessing را وارد می‌کند که برای نرمال‌سازی داده‌ها استفاده می‌شود.

from sklearn.preprocessing import StandardScaler

این خط یک شیء StandardScaler ایجاد می‌کند که برای نرمال‌سازی داده‌ها استفاده می‌شود.

scaler = StandardScaler()

در این بخش، داده‌های آموزشی با استفاده از fit\_transform نرمال‌سازی می‌شوند. این تابع ابتدا میانگین و انحراف معیار داده‌های آموزشی را محاسبه کرده و سپس داده‌ها را نرمال‌سازی می‌کند.

X\_train\_normalized = scaler.fit\_transform(X\_train)

در این بخش، داده‌های ارزیابی با استفاده از transform نرمال‌سازی می‌شوند. این تابع از میانگین و انحراف معیار محاسبه شده از داده‌های آموزشی استفاده کرده و داده‌های ارزیابی را نرمال‌سازی می‌کند.

X\_test\_normalized = scaler.transform(X\_test)

این بخش شکل داده‌های نرمال‌سازی شده برای مجموعه‌های آموزشی و ارزیابی را چاپ می‌کند تا اطمینان حاصل شود که داده‌ها به درستی نرمال‌سازی شده‌اند.

# چاپ شکل داده‌های نرمال‌سازی شده

print("Normalized Training data shape:", X\_train\_normalized.shape)

print("Normalized Test data shape:", X\_test\_normalized.shape)

پرسش2 بخش3:

این خطوط، کتابخانه‌های numpy برای عملیات‌های عددی، matplotlib.pyplot برای رسم نمودار و f1\_score و accuracy\_score از sklearn.metrics برای ارزیابی مدل را وارد می‌کنند.

import numpy as np

import matplotlib.pyplot as plt

from sklearn.metrics import f1\_score, accuracy\_score

این بخش، کلاس پرسپترون را تعریف می‌کند. در تابع \_\_init\_\_، پارامترهای نرخ یادگیری (learning\_rate) و تعداد تکرارها (n\_iter) تنظیم می‌شوند. همچنین وزن‌ها (weights)، بایاس (bias) و لیست اتلاف‌ها (losses) مقداردهی اولیه می‌شوند.

class Perceptron:

    def \_\_init\_\_(self, learning\_rate=0.1, n\_iter=100):

        self.learning\_rate = learning\_rate

        self.n\_iter = n\_iter

        self.weights = None

        self.bias = None

        self.losses = []

تابع fit برای آموزش مدل استفاده می‌شود. تعداد نمونه‌ها (n\_samples) و ویژگی‌ها (n\_features) از X استخراج می‌شود. وزن‌ها به صورت صفر مقداردهی اولیه می‌شوند و بایاس نیز صفر است. سپس مدل به تعداد n\_iter تکرار آموزش می‌بیند. در هر تکرار، برای هر نمونه:

خروجی خطی محاسبه می‌شود.

پیش‌بینی با استفاده از تابع فعال‌سازی انجام می‌شود.

خطای پیش‌بینی محاسبه و وزن‌ها و بایاس به‌روز‌رسانی می‌شوند.

مقدار اتلاف برای هر نمونه محاسبه و در لیست اتلاف‌ها ذخیره می‌شود.

def fit(self, X, y):

        n\_samples, n\_features = X.shape

        self.weights = np.zeros(n\_features)

        self.bias = 0

        for \_ in range(self.n\_iter):

            loss = 0

            for idx, x\_i in enumerate(X):

                linear\_output = np.dot(x\_i, self.weights) + self.bias

                y\_predicted = self.activation\_function(linear\_output)

                update = self.learning\_rate \* (y[idx] - y\_predicted)

                self.weights += update \* x\_i

                self.bias += update

                loss += (y[idx] - y\_predicted) \*\* 2

            self.losses.append(loss / n\_samples)

تابع predict برای پیش‌بینی خروجی مدل استفاده می‌شود. در این تابع، خروجی خطی محاسبه و با استفاده از تابع فعال‌سازی به مقدار باینری (0 یا 1) تبدیل می‌شود. تابع activation\_function نیز مقادیر خطی را به 0 و 1 تبدیل می‌کند.

 def predict(self, X):

        linear\_output = np.dot(X, self.weights) + self.bias

        y\_predicted = self.activation\_function(linear\_output)

        return y\_predicted

    def activation\_function(self, x):

        return np.where(x >= 0, 1, 0)

در این بخش، داده‌های مصنوعی تولید و به دو بخش آموزشی و تست تقسیم می‌شوند. داده‌های X شامل ویژگی‌ها و y شامل برچسب‌های هدف هستند. 8 نمونه اول برای آموزش و 2 نمونه آخر برای تست استفاده می‌شوند.

# تولید داده‌های مصنوعی برای نمونه

X = np.array([[2, 3], [1, 2], [3, 2], [0.5, 0.5], [1, 1], [2, 1], [1, 0.5], [2, 0.5], [1.5, 2.5], [1.5, 0.5]])

y = np.array([1, 1, 1, 0, 0, 0, 0, 0, 1, 0])

# تقسیم داده‌ها به آموزشی و تست

X\_train, X\_test = X[:8], X[8:]

y\_train, y\_test = y[:8], y[8:]

مدل پرسپترون با نرخ یادگیری 0.01 و 100 تکرار ایجاد و با داده‌های آموزشی آموزش داده می‌شود. سپس مدل با داده‌های تست ارزیابی شده و دقت (accuracy) و نمره F1 محاسبه می‌شوند.

p = Perceptron(learning\_rate=0.01, n\_iter=100)

p.fit(X\_train, y\_train)

# ارزیابی مدل

y\_pred = p.predict(X\_test)

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

f1 = f1\_score(y\_test, y\_pred)

print("Accuracy on test data:", accuracy)

print("F1-score on test data:", f1)

نمودار تابع اتلاف بر اساس تعداد تکرارها (epochs) رسم می‌شود. این نمودار نشان می‌دهد که چگونه مقدار اتلاف در هر تکرار کاهش یافته است.

p = Perceptron(learning\_rate=0.01, n\_iter=100)

p.fit(X\_train, y\_train)

# ارزیابی مدل

y\_pred = p.predict(X\_test)

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

f1 = f1\_score(y\_test, y\_pred)

print("Accuracy on test data:", accuracy)

print("F1-score on test data:", f1)

پرسش2 بخش3:

این خطوط، کتابخانه‌های numpy برای عملیات‌های عددی، matplotlib.pyplot برای رسم نمودار، SGDClassifier از sklearn.linear\_model برای ایجاد مدل پرسپترون، train\_test\_split از sklearn.model\_selection برای تقسیم داده‌ها و accuracy\_score و f1\_score از sklearn.metrics برای ارزیابی مدل را وارد می‌کنند.

import numpy as np

import matplotlib.pyplot as plt

from sklearn.linear\_model import SGDClassifier

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

from sklearn.metrics import accuracy\_score, f1\_score

در این بخش، داده‌های مصنوعی تولید می‌شوند. X شامل ویژگی‌ها و y شامل برچسب‌های هدف هستند.

X = np.array([[2, 3], [1, 2], [3, 2], [0.5, 0.5], [1, 1], [2, 1], [1, 0.5], [2, 0.5], [1.5, 2.5], [1.5, 0.5]])

y = np.array([1, 1, 1, 0, 0, 0, 0, 0, 1, 0])

این خط داده‌ها را به دو بخش آموزشی و تست تقسیم می‌کند. 20 درصد داده‌ها برای تست و 80 درصد برای آموزش استفاده می‌شوند. random\_state=42 برای اطمینان از تکرارپذیری نتایج استفاده می‌شود.

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

در این بخش، مدل پرسپترون با استفاده از SGDClassifier ایجاد می‌شود. پارامترهای مدل به شرح زیر هستند:

loss='hinge': استفاده از تابع اتلاف hinge (مناسب برای SVM).

learning\_rate='constant': استفاده از نرخ یادگیری ثابت.

eta0=0.01: مقدار اولیه نرخ یادگیری.

max\_iter=1000: حداکثر تعداد تکرارها.

tol=1e-3: مقدار تحمل برای معیار توقف.

random\_state=42: برای تکرارپذیری نتایج.

clf = SGDClassifier(loss='hinge', learning\_rate='constant', eta0=0.01, max\_iter=1000, tol=1e-3, random\_state=42)

clf.fit(X\_train, y\_train)

در این بخش، اگر مدل شامل نمودار تابع اتلاف (loss\_curve\_) باشد، آن را استخراج می‌کند. سپس مدل با داده‌های تست ارزیابی شده و دقت (accuracy) و نمره F1 (f1) محاسبه می‌شوند.

loss\_curve = clf.loss\_curve\_ if hasattr(clf, 'loss\_curve\_') else []

# ارزیابی مدل

y\_pred = clf.predict(X\_test)

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

f1 = f1\_score(y\_test, y\_pred)

print("Accuracy on test data:", accuracy)

print("F1-score on test data:", f1)

این بخش نمودار تابع اتلاف را بر اساس تعداد تکرارها رسم می‌کند، اگر داده‌های تابع اتلاف موجود باشد. در غیر این صورت، پیامی مبنی بر عدم وجود داده‌های تابع اتلاف چاپ می‌شود.

# رسم نمودار تابع اتلاف

if loss\_curve:

    plt.plot(loss\_curve, marker='o')

    plt.xlabel('Iteration')

    plt.ylabel('Loss')

    plt.title('Loss Curve over Iterations')

    plt.show()

else:

    print("No loss curve data available.")