MACHINE LEARNING: BÀI THỰC HÀNH PHẦN MÔ HÌNH PERCEPTRON.

Trong bài thực hành này chúng ta sẽ thực hiện mô hình Perceptron cùng với mô hình phân loại nhị phân Logistic Regression để so sánh. Trong một vài ví dụ chúng ta sẽ ôn lại việc sử dụng phương pháp PCA có thể để hiển thị dữ liệu và quan sát trực quan.

Ví dụ 1. Xây dựng mô hình Perceptron từ các thư viện cơ sở và thực hiện phân loại dữ liệu nhân tạo để minh họa. Trong ví dụ này, chúng ta sẽ tạo một tập dữ liệu ngẫu nhiên 02 chiều, với phân bố chuẩn có kỳ vọng là các điểm tương đối biệt lập và ma trận hiệp phương sai phù hợp để đảm bảo dữ liệu hầu như tách được tuyến tính. Dưới đây là đoạn chương trình liên kết các thư viện và tạo dữ liệu:

```
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(2)

means = [[2, 2], [4, 2]]
cov = [[.3, .2], [.2, .3]]
N = 30
X0 = np.random.multivariate_normal(means[0], cov, N).T
X1 = np.random.multivariate_normal(means[1], cov, N).T


X = np.concatenate((X0, X1), axis = 1)
y = np.concatenate((np.ones((1, N)), -1*np.ones((1, N))), axis = 1)
# Xbar
X = np.concatenate((np.ones((1, 2*N)), X), axis = 0)
```

Hiển thị dữ liệu để quan sát

```
fig, ax = plt.subplots(figsize=(5, 5))

ani = plt.cla()
#plot points
ani = plt.plot(X0[0, :], X0[1, :], 'b^', markersize = 8, alpha = .8)
ani = plt.plot(X1[0, :], X1[1, :], 'ro', markersize = 8, alpha = .8)
ani = plt.axis([0 , 6, -2, 4])
plt.show()
```

Xây dựng hàm $y = h_w(x) = w_0 + w^T x = \ddot{w}^T \ddot{x}$

```
# Define h_w(x):= W^T.x + w_0 = \bar{W}^T . \bar{x}
def h(w, x):
    return np.sign(np.dot(w.T, x))
```

Hàm kiểm tra điều kiện dừng

```
#Stop condition
def has_converged(X, y, w):
    return np.array_equal(h(w, X), y) #True if h(w, X) == y else False
```

Vòng lặp chính tìm đường phân chia (bộ hệ số W) theo phương pháp Gradient Descent

```
def perceptron(X, y, w_init):
    w = [w_init]
    N = X.shape[1]
    mis_points = [] # set of miss position points
```

```
    while True:
        # mix data
        mix_id = np.random.permutation(N)
        for i in range(N):
            xi = X[:, mix_id[i]].reshape(3, 1)
            yi = y[0, mix_id[i]]
            if h(w[-1], xi)[0] != yi:
                mis_points.append(mix_id[i])
                w_new = w[-1] + yi*xi

                w.append(w_new)

        if has_converged(X, y, w[-1]):
            break
    return (w, mis_points)
```

Gọi hàm lặp và in ra kết quả là bộ trọng số W ở vòng lặp cuối

```
d = X.shape[0]
w_init = np.random.randn(d, 1)
(w, m) = perceptron(X, y, w_init)
print(w[-1])
```

Tiếp theo chúng ta xây dựng phương thức vẽ đường phân chia để quan sát kết quả

```
def draw_line(w):
    w0, w1, w2 = w[0], w[1], w[2]
    if w2 != 0:
        x11, x12 = -100, 100
        return plt.plot([x11, x12], [-(w1*x11 + w0)/w2, -(w1*x12 + w0)/w2], 'k')
    else:
        x10 = -w0/w1
        return plt.plot([x10, x10], [-100, 100], 'k')
```

Trong phần cuối, chúng ta sẽ hiển thị kết quả dạng ảnh động (gif) để quan sát sự thay đổi của đường phân chia qua từng bước lặp. Tên ảnh kết quả sẽ là 'pla_vis.gif', trong chương trình này ảnh được lưu vào ổ D:\. Các bạn sửa đường dẫn. Khi chương trình kết thúc đường phân chia cuối cùng sẽ được vẽ ra. Chúng ta sử dụng thư viện matplotlib.animation để tạo gif.

```
## Visualization
import matplotlib.animation as animation
from matplotlib.animation import FuncAnimation

def viz_alg_1d_2(w):
    it = len(w)
    fig, ax = plt.subplots(figsize=(5, 5))

    def update(i):
        ani = plt.cla()
        #points
        ani = plt.plot(X0[0, :], X0[1, :], 'b^', markersize = 8, alpha = .8)
        ani = plt.plot(X1[0, :], X1[1, :], 'ro', markersize = 8, alpha = .8)
        ani = plt.axis([0 , 6, -2, 4])
        i2 =  i if i < it else it-1
        ani = draw_line(w[i2])
        if i < it-1:
            # draw one  misclassified point
            circle = plt.Circle((X[1, m[i]], X[2, m[i]]), 0.15, color='k', fill =
False)
            ax.add_artist(circle)
        # hide axis
```

```
        cur_axes = plt.gca()
        cur_axes.axes.get_xaxis().set_ticks([])
        cur_axes.axes.get_yaxis().set_ticks([])

        label = 'PLA: iter %d/%d' %(i2, it-1)
        ax.set_xlabel(label)
        return ani, ax

    anim = FuncAnimation(fig, update, frames=np.arange(0, it + 2), interval=1000)
    # save
    anim.save('D:\\pla_vis.gif', dpi = 100, writer = 'imagemagick')
    plt.show()

viz_alg_1d_2(w)
```

Ví dụ 2. Trong ví dụ dưới đây chúng ta sử dụng dữ liệu sóng thủy âm Sonar. Tệp dữ liệu có 60 cột ứng với 60 thuộc tính (trường) không có tiêu đề, là tham số của các mẫu sóng âm phản hồi; cột thứ 61 là đầu ra phân loại (y), với ký tự "R" nghĩa là Rock; ký tự "M" nghĩa là Mine (vật thể kim loại hình trụ). Toàn bộ tệp có 208 bản ghi. Thông tin thêm về dữ liệu có thể tìm hiểu tại link https://archive.ics.uci.edu/ml/datasets/Connectionist+Bench+(Sonar,+Mines+vs.+Rocks) hoặc có thể lấy trong tệp Sonar.all-data.csv đính kèm.

Trong ví dụ này chúng ta sử dụng thư viện matplotlib và seaborn để thực hiện việc hiển thị trực quan cấu trúc dữ liệu (tham khảo lại bài thực hành phần PCA).

Phần liên kết thư viện

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from matplotlib import pyplot
import matplotlib.pyplot as plt
import seaborn as sns
```

Tiếp theo, chúng ta sẽ đọc toàn bộ dữ liệu. Các bạn cần sửa lại đường dẫn đến tệp dữ liệu tương ứng trên máy của các bạn

```
main_df = pd.read_csv('D:\\Teach_n_Train\\Machine Learning\\code\\data\\sonar.all-data.csv',header=None)
main_df
```

Chúng ta sẽ kiểm tra độ cân bằng của dữ liệu bằng cách nhóm theo cột cuối cùng (chỉ số cột là 60 – tính từ không), tức là cột đầu ra y:

```
main_df[60].value_counts().plot(kind='barh')
```

Đoạn chương trình dưới đây đổi nhãn của đầu ra y từ {'M', 'R'} sang {1, -1} (tức là M ứng với class 1; R ứng với class -1)

```
y_df = main_df[60]
targes_label = {'M': 1,'R': -1}
targes_df = [targes_label[item] for item in y_df]
print(targes_df)
```

Lấy phần dữ liệu quan sát (bỏ cột 60)

```
inputs_df = main_df.drop(60, axis=1)
```

Sau đó bổ sung cột thêm trường dữ liệu $x_0 = 1$ vào bên trái

```
inputs_df = main_df.drop(60, axis=1)
x0 = np.ones(( inputs_df.shape[0], 1))
X = np.concatenate((x0, inputs_df), axis = 1)
```

Số chiều dữ liệu hiện tại sẽ là 61 (trường) x 208 (bản ghi). Chúng ta chia dữ liệu thành tập train : test với tỷ lệ là 7:3

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, targes_df, test_size=0.30,
random_state=42)
```

Bài tập tự thực hành:

Tiếp theo, các bạn hãy sử dụng các hàm đã có ở Ví dụ 1, sau đó huấn luyện mô hình bằng tập dữ liệu X_train, y_train. Chú ý chúng ta cần bỏ qua các phần lệnh phục vụ việc hiển thị kết quả dạng hình vẽ.

Sau đó, hãy viết bổ sung phần chương trình thực hiện việc chạy test với bộ hệ số đã có, tham khảo các bài phần trước và tính độ chính xác bằng accuracy, recall và và precision.

Ví dụ 3. Xét tập dữ liệu phân loại bệnh nhân ung thư vú của Đại học Wisconsin–Madison, Hoa Kỳ. Dữ liệu có sẵn trong thư viện sklearn.

Dữ liệu có 569 bản ghi (mẫu), với 30 thuộc tính. Bệnh nhân được chia làm hai loại: u lành tính (B – Begnin) có 357 mẫu và u ác tính (M – Malignant) có 212 mẫu.

```
from sklearn import datasets

cancer_data = datasets.load_breast_cancer()

# show to test record 5th
print(cancer_data.data[5])

print(cancer_data.data.shape)
#target set
print(cancer_data.target)

from sklearn.model_selection import train_test_split

cancer_data = datasets.load_breast_cancer()

X_train, X_test, y_train, y_test = train_test_split(cancer_data.data,
                                    cancer_data.target,
                                    test_size=0.3,random_state=109)
```

Bài tập tự thực hành: Thực hiện các bước tương tự như yêu cầu của Ví dụ 2. Sử dụng lại dữ liệu trong ví dụ 2 và ví dụ 3, sau đó

- Thực hiện giảm số chiều về 02 chiều
- Hiển thị kết quả dữ liệu đã giảm chiều lên màn hình, với 2 phân lớp được biểu diễn bằng 2 màu khác nhau.

- Thực hiện phân loại bằng hồi quy Logistic và so sánh kết quả với phương pháp Perceptron đã thực hiện ở phần trước.

Ví dụ 4 (Bài tập tự thực hành – Nộp trong buổi thực hành). Chúng ta sử dụng lại dữ liệu chứa các thông tin về nhân khẩu, hành vi thói quen và lịch sử bệnh lý để xác định khả năng bị trụy tim ở một bệnh nhân đã cho trong phần thực hành hồi quy Logistic. Mục tiêu phân loại là dự đoán liệu bệnh nhân có nguy cơ mắc bệnh tim mạch vành (CHD) trong tương lai (10 năm) hay không. Bộ dữ liệu cung cấp thông tin của bệnh nhân, có thể tìm thấy tại link https://www.kaggle.com/amanajmera1/framingham-heart-study-dataset/data hoặc lấy từ tệp đính kèm framingham.csv trong bài thực hành hồi quy Logistic.

- Hãy xử lý sơ bộ dữ liệu như trong yêu cầu của phần hồi quy logistic. Chia tập dữ liệu thành Traning:Validation = 7:3.
- Thực hiện phân loại bằng phương pháp Perceptron. Tính các độ đo Accuracy, Precision và Recall để đánh giá kết quả.
- Thực hiện phân loại bằng phương pháp Hồi quy Logistic trên cùng bộ dữ liệu training:validation đã có ở ý trên. Tính các độ đo Accuracy, Precision và Recall và so sánh kết quả.
- Hãy giải thích về kết quả thu được của các mô hình cũng như nhận xét trên độ chính xác của mỗi mô hình.