Week6_Perceptron_ViDu4

Phạm Ngọc Hải

March 25, 2024

1 Ví dụ 1.

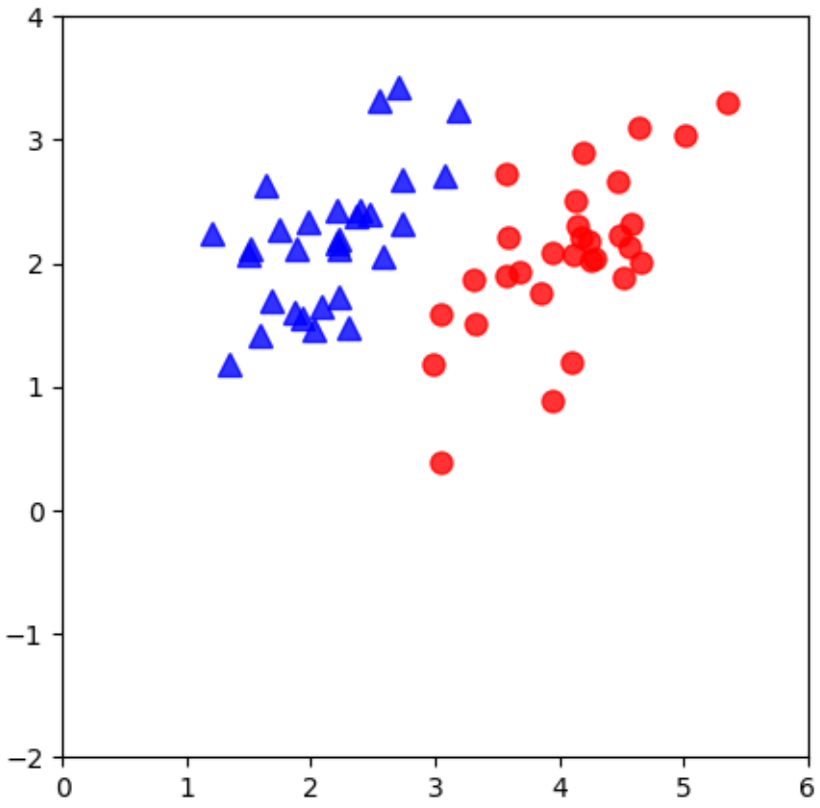
Ví dụ về xây dựng mô hình Perceptron

Khởi tạo dữ liệu 2D

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(2)

means = [[2, 2], [4, 2]]
cov = [[0.3, 0.2], [0.2, 0.3]]
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 kết quả

[ ]:
```
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()
```



1.1 Tự xây dựng hàm

Xây dựng hàm $y = h_w(x) = w_0 + w^T * x$

[ ]:
```
# 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))
```

Xây dựng 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
```

Xây dựng hàm perceptron tìm 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 và in ra 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])
```

```
[[ 13.97858527]
 [-15.02901929]
 [ 14.38743059]]
```

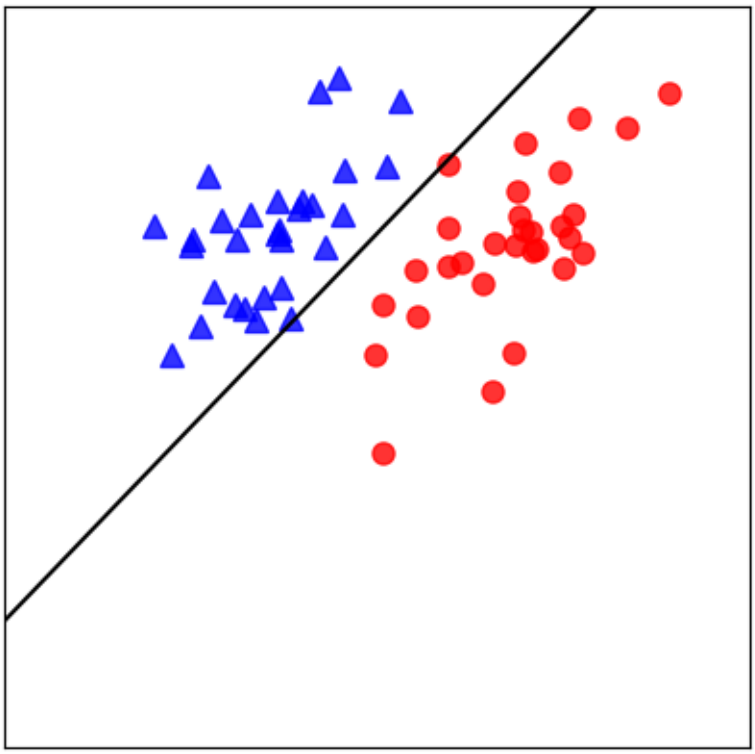
Xây dựng hàm vẽ đường phân chia quan sát kết quả

[ ]:
```
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")
```

Sử dụng hình ảnh động GIF để theo dõi kết quả tìm W qua các bước lặp. Tên ảnh kết quả là pla_vis.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=0.8)
        ani = plt.plot(X1[0, :], X1[1, :], "ro", markersize=8, alpha=0.8)
        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("W6_pla_vis.gif", dpi=100, writer="imagemagick")
    plt.show()


viz_alg_1d_2(w)
```



PLA: iter 75/75

1.2 Sử dụng thư viện

Init: Perceptron() trong thư viện Fit: .fit(X_train, y_train) Predict: .predict(X_validation) Hệ số: .coef_

2 Ví dụ 4 (Bài tập tự thực hành - Nộp trong buổi thực hành)

So sánh thực hiện phương pháp *Perceptron* và *Hồi quy Logistic* trên tập dữ liệu nguy cơ mắc bệnh tim mạch vành (CHD) trong 10 năm tới

2.1 Xử lý sơ bộ data như trong bài Hồi quy Logistic.

Đọc dữ liệu

[ ]:
```
import pandas as pd
df = pd.read_csv("data/framingham.csv")
df.head()
df.isnull().sum()
```

[ ]:
```
male               0
age                0
education        105
```

```
currentSmoker        0
cigsPerDay          29
BPMeds              53
prevalentStroke      0
prevalentHyp         0
diabetes             0
totChol             50
sysBP                0
diaBP                0
BMI                 19
heartRate            1
glucose            388
TenYearCHD           0
dtype: int64
```

Loại bỏ dữ liệu N/A trong dữ liệu

[ ]:
```
df = df.dropna(how="any", axis=0)
```

Nhìn qua về dữ liệu

[ ]:
```
print(df[:3])
```

```
   male  age  education  currentSmoker  cigsPerDay  BPMeds  prevalentStroke  \
0     1   39        4.0              0         0.0     0.0                0
1     0   46        2.0              0         0.0     0.0                0
2     1   48        1.0              1        20.0     0.0                0

   prevalentHyp  diabetes  totChol  sysBP  diaBP    BMI  heartRate  glucose  \
0             0         0    195.0  106.0   70.0  26.97       80.0     77.0
1             0         0    250.0  121.0   81.0  28.73       95.0     76.0
2             0         0    245.0  127.5   80.0  25.34       75.0     70.0

   TenYearCHD
0           0
1           0
2           0
```

Phân chia Train:Validation = 7:3

[ ]:
```
from sklearn.model_selection import train_test_split

# Chia dataframe thành features (X) và target variable (y)
vd4_X = df.drop("TenYearCHD", axis=1)
vd4_y = df["TenYearCHD"]

# Chia dữ liệu thành tập train và tập validation theo tỷ lệ 7:3
vd4_X_train, vd4_X_valid, vd4_y_train, vd4_y_valid = train_test_split(
    vd4_X, vd4_y, test_size=0.3, random_state=42
```

```
)

# Xem số lượng bản ghi trong mỗi tập
print("Số lượng bản ghi trong tập train:", len(vd4_X_train))
print("Số lượng bản ghi trong tập validation:", len(vd4_X_valid))
```

```
Số lượng bản ghi trong tập train: 2559
Số lượng bản ghi trong tập validation: 1097
```

2.2 Thực hiện phân loại bằng phương pháp Perceptron.

Tính các độ đo Accuracy, Precision và Recall để đánh giá kết quả.

[ ]:
```
from sklearn.linear_model import Perceptron

# Khởi tạo mô hình hồi quy logistic
vd4_perceptron_model = Perceptron()
# Huấn luyện mô hình trên tập train
vd4_perceptron_model.fit(vd4_X_train, vd4_y_train)
```

[ ]:
```
Perceptron()
```

[ ]:
```
# Dự đoán trên tập validation
vd4_y_pred_valid_perceptron = vd4_perceptron_model.predict(vd4_X_valid)
print(vd4_y_pred_valid_perceptron[:5])
```

```
[1 1 1 0 1]
```

[ ]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score

# Tính toán accuracy
vd4_accuracy_perceptron = accuracy_score(vd4_y_valid,␣
↪vd4_y_pred_valid_perceptron)

# Tính toán precision
vd4_precision_perceptron = precision_score(vd4_y_valid,␣
↪vd4_y_pred_valid_perceptron)

# Tính toán recall
vd4_recall_perceptron = recall_score(vd4_y_valid, vd4_y_pred_valid_perceptron)

print("Accuracy:", vd4_accuracy_perceptron)
print("Precision:", vd4_precision_perceptron)
print("Recall:", vd4_recall_perceptron)
```

```
Accuracy: 0.24703737465815861
Precision: 0.16322314049586778
Recall: 0.9080459770114943
```

2.3 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.

[ ]:
```
from sklearn.linear_model import LogisticRegression

# Khởi tạo mô hình hồi quy logistic
vd4_logreg_model = LogisticRegression()
# Huấn luyện mô hình trên tập train
vd4_logreg_model.fit(vd4_X_train, vd4_y_train)
```

```
/home/harito/venv/py/lib/python3.11/site-
packages/sklearn/linear_model/_logistic.py:444: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

[ ]:
```
LogisticRegression()
```

[ ]:
```
# Dự đoán trên tập validation
vd4_y_pred_valid_logreg = vd4_logreg_model.predict(vd4_X_valid)
print(vd4_y_pred_valid_logreg[:5])
```

```
[0 0 0 0 0]
```

[ ]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score

# Tính toán accuracy
vd4_accuracy_logreg = accuracy_score(vd4_y_valid, vd4_y_pred_valid_logreg)

# Tính toán precision
vd4_precision_logreg = precision_score(vd4_y_valid, vd4_y_pred_valid_logreg)

# Tính toán recall
vd4_recall_logreg = recall_score(vd4_y_valid, vd4_y_pred_valid_logreg)

print("Accuracy:", vd4_accuracy_logreg)
print("Precision:", vd4_precision_logreg)
print("Recall:", vd4_recall_logreg)
```

```
Accuracy: 0.8413855970829535
Precision: 0.5
Recall: 0.034482758620689655
```

2.4 Hãy giải thích về kết quả thu được của các mô hình cũng như nhận xét trên độ chính xác của mỗi mô hình.

Phương pháp Perceptron:

- Accuracy: 0.247 - Tỷ lệ dự đoán chính xác trên tổng số mẫu.
- Precision: 0.163 - Tỉ lệ số điểm dự đoán dương thực tế chính xác so với tổng số điểm dự đoán dương.
- Recall: 0.908 - Tỉ lệ số điểm dự đoán dương thực tế chính xác so với tổng số điểm thực tế dương.

Với Perceptron, chúng ta thấy rằng độ chính xác (accuracy) thấp, chỉ khoảng 24.7%, trong khi độ chính xác của precision (độ chính xác của các dự đoán dương) thấp (chỉ khoảng 16.3%), và recall (tỉ lệ các điểm dương thực tế được dự đoán đúng) cao (khoảng 90.8%). Kết quả này cho thấy mô hình có xu hướng dự đoán sai nhiều mẫu âm tính và dự đoán đúng nhiều mẫu dương tính.

Phương pháp Logistic Regression:

- Accuracy: 0.841 - Tỷ lệ dự đoán chính xác trên tổng số mẫu.
- Precision: 0.5 - Tỉ lệ số điểm dự đoán dương thực tế chính xác so với tổng số điểm dự đoán dương.
- Recall: 0.034 - Tỉ lệ số điểm dự đoán dương thực tế chính xác so với tổng số điểm thực tế dương.

Với Logistic Regression, chúng ta thấy rằng độ chính xác (accuracy) cao, đạt khoảng 84.1%, nhưng precision (độ chính xác của các dự đoán dương) thấp (chỉ khoảng 50%), và recall (tỉ lệ các điểm dương thực tế được dự đoán đúng) thấp (khoảng 3.4%). Kết quả này cho thấy mô hình có xu hướng dự đoán đúng nhiều mẫu âm tính, nhưng lại dự đoán sai nhiều mẫu dương tính.

Tóm lại, dựa trên các chỉ số đánh giá, mô hình Logistic Regression có độ chính xác cao hơn so với Perceptron. Tuy nhiên, cả hai mô hình đều có thể cần được cải thiện, đặc biệt là trong việc dự đoán các mẫu dương tính.