

Ứng dụng Trí tuệ nhân tạo trong Nuôi trồng thủy sản

NGUYỄN HẢI TRIỀU¹

¹ Bộ môn Kỹ thuật phần mềm,
Khoa Công nghệ thông tin, Trường ĐH Nha Trang

NhaTrang, September 2024

1 Using Logistic Regression for Classification

- Single Layer Neural Networks
- Activation Function
- Logistic regression loss function
- Model Training with Stochastic Gradient Descent
- Automatic Differentiation in PyTorch
- The PyTorch API
- Training a Logistic Regression Model in PyTorch
- Feature Normalization

Training a Logistic Regression Model I

Trong phần này, chúng ta sẽ tìm hiểu cách **huấn luyện mô hình hồi quy logistic trong PyTorch**. Chúng ta sẽ sử dụng lại code và bộ dữ liệu từ chương 2 (chi tiết về code xem file **Chuong_3.3_code**)

```
1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 df = pd.read_csv("perceptron_toydata-truncated.txt", sep="\t")
6 X_train = df[["x1", "x2"]].values
7 y_train = df["label"].values
8 np.bincount(y_train) # kiểm tra phân bố số lượng nhãn của lớp 0, 1.
9 # vẽ phân bố các điểm dữ liệu đầu vào với 2 đặc trưng
10 plt.plot(
11     X_train[y_train == 0, 0],
12     X_train[y_train == 0, 1],
13     marker="D",
```

Training a Logistic Regression Model II

```
14     markersize=10,
15     linestyle="",
16     label="Class 0",
17 )
18
19 plt.plot(
20     X_train[y_train == 1, 0],
21     X_train[y_train == 1, 1],
22     marker="^",
23     markersize=13,
24     linestyle="",
25     label="Class 1",
26 )
27
28 plt.legend(loc=2)
29
30 plt.xlim([-5, 5])
31 plt.ylim([-5, 5])
32
33 plt.xlabel("Feature  $x_1$ ", fontsize=12)
```

Training a Logistic Regression Model III

```
34 plt.ylabel("Feature $x_2$", fontsize=12)
35
36 plt.grid()
37 plt.show()
38
39 # chuẩn hoá dữ liệu đầu vào
40 X_train = (X_train - X_train.mean(axis=0)) / X_train.std(axis=0)
```

Chuẩn hoá dữ liệu

We can now see that the data set features are clustered around zero or they are centered at zero. And also the spread is a little bit smaller here. This is an important step to make the logistic regression training smoother.

Cài đặt mô hình hồi quy logistic trong PyTorch

Đầu tiên chúng ta tạo lớp *LogisticRegression* kế thừa từ *torch.nn.Module*, sử dụng lớp *Linear* thay cho cho việc khởi tạo trọng số như mạng Perceptron. Tiếp theo định nghĩa phương thức *forward* để tính giá trị đầu ra của lớp *Linear* và hàm kích hoạt *sigmoid*.

```
1 import torch
2
3 class LogisticRegression(torch.nn.Module):
4
5     def __init__(self, num_features):
6         super().__init__()
7         self.linear = torch.nn.Linear(num_features, 1)
8
9     def forward(self, x):
10         logits = self.linear(x)
11         probas = torch.sigmoid(logits)
12         return probas
```

Khởi tạo một ví dụ và xem kết quả dự đoán

Khởi tạo mô hình Logistic regression với một mẫu huấn luyện có 2 đặc trưng $x = \text{torch.tensor}([1.1, 2.1])$

```
1 torch.manual_seed(1)
2 model = LogisticRegression(num_features=2)
3 x = torch.tensor([1.1, 2.1])
4 with torch.no_grad():
5     proba = model(x)
6
7 print(proba)
```

Trong đó, *with torch.no_grad()* có nghĩa là quá trình dự đoán sẽ không liên quan đến sơ đồ tính toán gradient mà PyTorch thực hiện. Điều này giúp tiết kiệm bộ nhớ và giúp cho mô hình chạy dự đoán nhanh hơn.

Data loader

Bỏ qua ví dụ chỉ có 1 mẫu huấn luyện ở trước, ta đi đến phần chuẩn bị *Data loader* là dữ liệu dùng để huấn luyện mô hình.

```
1  from torch.utils.data import Dataset, DataLoader
2
3  class MyDataset(Dataset):
4      def __init__(self, X, y):
5          self.features = torch.tensor(X, dtype=torch.float32)
6          self.labels = torch.tensor(y, dtype=torch.float32)
7
8      def __getitem__(self, index):
9          x = self.features[index]
10         y = self.labels[index]
11         return x, y
12
13     def __len__(self):
14         return self.labels.shape[0]
15
16 train_ds = MyDataset(X_train, y_train)
17
```



```
18 train_loader = DataLoader(  
19     dataset=train_ds,  
20     batch_size=10,  
21     shuffle=True,  
22 )
```

MyDataset

Chúng ta sẽ định nghĩa lớp dữ liệu huấn luyện của mình *MyDataset* dựa trên 2 lớp *Dataset*, *DataLoader* từ thư viện *torch.utils.data*. Với đầu vào là tensor đặc trưng X_{train} và tensor label y_{train} tương ứng, ta định nghĩa phương thức `__getitem__` để lấy từng mẫu dữ liệu huấn luyện và nhận thông qua *index*. Phương thức `__len__` để lấy số lượng mẫu có trong bộ dữ liệu huấn luyện.

Dataloader

Để sử dụng *thuật toán huấn luyện minibatch stochastic gradient descent*, bộ dữ liệu huấn luyện cần chia thành các *batches*, ở đây chúng ta sử dụng lớp *Dataloader* với thuộc tính *batch_size* của *PyTorch*. Bên cạnh đó ta cũng xáo trộn bộ dữ liệu *shuffle=True* trước mỗi epoch để đảm bảo quá trình học được nhanh hơn.

Huấn luyện mô hình I

Quá trình huấn luyện mô hình Logistic regression, ta khởi tạo mô hình với lớp *LogisticRegression(num_features=2)* và thuật toán huấn luyện cập nhật tham số mô hình là *torch.optim.SGD(model.parameters(), lr=0.05)*. Chúng ta sẽ huấn luyện trong vòng *20 epochs* và *learning rate = 0.05*. Các tham số này có thể thay đổi tùy thực nghiệm.

```
1 import torch.nn.functional as F
2
3 torch.manual_seed(1)
4 model = LogisticRegression(num_features=2)
5 optimizer = torch.optim.SGD(model.parameters(), lr=0.05)
6 num_epochs = 20
7 for epoch in range(num_epochs):
8
9     model = model.train()
```

Huấn luyện mô hình II

```
10 for batch_idx, (features, class_labels) in enumerate(train_loader):
11
12     probas = model(features)
13
14     loss = F.binary_cross_entropy(probas, class_labels.view(probas.shape))
15
16     optimizer.zero_grad()
17     loss.backward()
18     optimizer.step()
19
20     ### LOGGING
21     print(f'Epoch: {epoch+1:03d}/{num_epochs:03d}'
22           f' | Batch {batch_idx:03d}/{len(train_loader):03d}'
23           f' | Loss: {loss:.2f}')
```

Huấn luyện mô hình III

Khi lặp qua mỗi epoch, nên thiết lập chế độ *model.train()*.

Tiếp theo, *for batch_idx, (features, class_labels) in enumerate(train_loader)*: sẽ giúp chúng ta duyệt qua các *mini batch* trong *train_loader*, ứng với mỗi *mini batch* ta có các số lượng mẫu huấn luyện tương ứng (*features, class_labels*). Trong phần *DataLoader* trước, ta chọn *batch_size = 10*, tức là mỗi *mini batch* sẽ có 10 dòng dữ liệu mẫu huấn luyện. Với bộ dữ liệu có 20 dữ liệu huấn luyện, PyTorch sẽ chia ra được 2 *mini batches*, *batch_idx = {0, 1}*. Dĩ nhiên, *batch_size* là siêu tham số, chúng ta có thể điều chỉnh dựa vào *learning experience*.

Huấn luyện mô hình IV

Ở vòng lặp bên trong,

```
1 probas = model(features)
2 loss = F.binary_cross_entropy(probas, class_labels.view(probas.shape))
```

chúng ta thu được *class membership probabilities* *probas* cho các training example trong mini batch. Khi tính *loss* ta **thu được một giá trị tương ứng tương ứng cho một mini batch**.

Lưu ý: để tính được loss thì true label class (hiện tại `torch.Size([10])`) phải cùng kích thước với *probas* (`torch.Size([10, 1])`), do đó ta gọi `class_labels.view(probas.shape)` để quy về cùng 1 kích thước tensor.

Huấn luyện mô hình V

Tiếp theo, chúng ta tiến hành cập nhật trọng số bằng thuật toán *minibatch stochastic gradient descent*:

```
1 optimizer.zero_grad() # tránh cộng dồn gradient ở batch trước  
2 loss.backward() # tính đạo hàm riêng theo các tham số mô hình  
3 optimizer.step() # cập nhật trọng số để cực tiểu loss.
```

Huấn luyện mô hình VI

Kết quả huấn luyện

Dựa vào kết quả, ta thấy rằng **hàm Loss giảm nhỏ dần** so với ban đầu và đủ nhỏ để nhận thấy rằng *quá trình huấn luyện mô hình Logistic regression có hiệu quả.*

```
1 Epoch: 001/020 | Batch 000/002 | Loss: 0.67
2 Epoch: 001/020 | Batch 001/002 | Loss: 0.73
3 Epoch: 002/020 | Batch 000/002 | Loss: 0.67
4 Epoch: 002/020 | Batch 001/002 | Loss: 0.67
5 Epoch: 003/020 | Batch 000/002 | Loss: 0.60
6 ...
7 Epoch: 019/020 | Batch 000/002 | Loss: 0.34
8 Epoch: 019/020 | Batch 001/002 | Loss: 0.39
9 Epoch: 020/020 | Batch 000/002 | Loss: 0.33
10 Epoch: 020/020 | Batch 001/002 | Loss: 0.38
```


Ước lượng hiệu quả của mô hình I

Để đánh giá mô hình sau khi huấn luyện có dự đoán tốt không, chúng ta có thể sử dụng hàm “ *compute_accuracy*”

```
1 def compute_accuracy(model, dataloader):
2
3     model = model.eval()
4     correct = 0.0
5     total_examples = 0
6
7     for idx, (features, class_labels) in enumerate(dataloader):
8         with torch.no_grad():
9             probas = model(features)
10
11         pred = torch.where(probas > 0.5, 1, 0)
12         lab = class_labels.view(pred.shape).to(pred.dtype)
13
14         compare = lab == pred
15         correct += torch.sum(compare)
```

Ước lượng hiệu quả của mô hình II

```
16         total_examples += len(compare)
17
18     return correct / total_examples
19 train_acc = compute_accuracy(model, train_loader)
20 print(f"Accuracy: {train_acc*100}%")
```

evaluation mode

Trước khi tính đánh giá hiệu quả huấn luyện mô hình, **chúng ta cần đặt model vào evaluation mode: `model.eval()`**. Điều này giúp mô hình *tách rời với sơ đồ tính toán gradient* và sẽ thực hiện *tính toán khác so với huấn luyện ở các lớp `BatchNorm`, `BatchNorm` ...* được nhắc ở chương sau.

Ước lượng hiệu quả của mô hình III

```
torch.where(probas > 0.5, 1, 0)
```

Nhắc lại rằng *probas* hiện tại đang ở dạng xác suất thành phần của class, chúng ta cần hàm threshold để phân lớp dữ liệu mẫu thuộc 0 hoặc 1 dựa vào ngưỡng 0.5. Vậy nên chúng ta sử dụng *torch.where(probas > 0.5, 1, 0)*.

```
compare = lab == pred
```

Sử dụng *compare* chúng ta có thể tính được số lượng nhãn dữ liệu đoán đúng với nhãn thật sự của các điểm dữ liệu trong mini batch bằng cách *correct += torch.sum(compare)*.

The Problem with Features on Different Scales

Ở phần huấn luyện mạng Logistic regression, chúng ta đã thực hiện chuẩn hoá dữ liệu đầu vào. *Câu hỏi đặt ra tại sao cần phải chuẩn hoá?* Hãy xem bộ dữ liệu thực tế [Wine dataset](#) với mong muốn dự đoán chất lượng của rượu từ 13 thông số đặc trưng khác nhau.

	Alcohol	Malic acid	Ash	Alcalinity of ash	Magnesium	Total phenols	Flavanoids	Nonflavanoid phenols	Proanthocyanins	Color intensity	Hue	OD280/OD315 of diluted wines	Proline
0	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065
1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050
2	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185
3	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480
4	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735
...
173	13.71	5.65	2.45	20.5	95	1.68	0.61	0.52	1.06	7.70	0.64	1.74	740
174	13.40	3.91	2.48	23.0	102	1.80	0.75	0.43	1.41	7.30	0.70	1.56	750
175	13.27	4.28	2.26	20.0	120	1.59	0.69	0.43	1.35	10.20	0.59	1.56	835
176	13.17	2.59	2.37	20.0	120	1.65	0.68	0.53	1.46	9.30	0.60	1.62	840
177	14.13	4.10	2.74	24.5	96	2.05	0.76	0.56	1.35	9.20	0.61	1.60	560

178 rows x 13 columns

The Problem with Features on Different Scales

Dễ dàng quan sát được rằng các đặc trưng chênh lệch rất lớn.

Điều này ảnh hưởng tiêu cực đến thuật toán học SGD

	Alcohol	Malic acid	Ash	Alcalinity of ash	Magnesium	Total phenols	Flavanoids	Nonflavanoid phenols	Proanthocyanins	Color intensity	Hue	OD280/OD315 of diluted wines	Proline
0	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065
1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050
2	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185
3	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480
4	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735
...
173	13.71	5.65	2.45	20.5	95	1.68	0.61	0.52	1.06	7.70	0.64	1.74	740
174	13.40	3.91	2.48	23.0	102	1.80	0.75	0.43	1.41	7.30	0.70	1.56	750
175	13.27	4.28	2.26	20.0	120	1.59	0.69	0.43	1.35	10.20	0.59	1.56	835
176	13.17	2.59	2.37	20.0	120	1.65	0.68	0.53	1.46	9.30	0.60	1.62	840
177	14.13	4.10	2.74	24.5	96	2.05	0.76	0.56	1.35	9.20	0.61	1.60	560

178 rows x 13 columns

The Problem with Features on Different Scales

Để hiểu lý do tại sao, ta xem lại công thức cập nhật trọng số của SGD

$$\mathbf{w} = \mathbf{w} + \alpha \nabla L_{\mathbf{w}}.$$

Trong đó, $\nabla L_{\mathbf{w}} = [\partial L / \partial w_1, \partial L / \partial w_2, \dots, \partial L / \partial w_m]^T$. *Rõ ràng, $\partial L / \partial w_1, \dots, \partial L / \partial w_m$ phụ thuộc vào các đặc trưng đầu vào. Vậy nên, khi các đặc trưng có tỉ lệ chênh lệch lớn thì các hệ số $\partial L / \partial w_i$ cũng chênh lệch lớn.*

Do đó, về mặt tổng thể chúng ta sẽ *không thể tìm được một **learning rate** α nào có thể thoả mãn được hết **tất cả các hệ số chênh lệch đó**.*

The Problem with Features on Different Scales

Main Advantages of Normalized Features

Nếu đặc trưng đầu vào được chuẩn hoá, chúng ta sẽ thu được:

- ➊ Easier to find a good learning rate
- ➋ Getting numerically more stable gradients
- ➌ The training will be faster due to faster convergence, which essentially means that we need fewer training epochs.

Common Feature Normalization Techniques

Hiện nay, chúng ta thường sử dụng 2 kỹ thuật chuẩn hoá dữ liệu trong ML/DL cho dữ liệu có cấu trúc:

- 1 0-1 Normalization/Min-max normalization
- 2 Z-Score Standardization

0-1 Normalization/Min-max normalization

Now one common normalization technique is **0-1 normalization** or sometimes also called **min-max normalization**:

$$\text{norm}(x_j^{[i]}) = \frac{x_j^{[i]} - \min(x_j)}{\max(x_j) - \min(x_j)}$$

Để hiểu hơn công thức, ta xem ví dụ về Wine dataset trước đó:

	Alcohol	Malic acid	Ash	Alcalinity of ash	Magnesium	Total phenols	Flavanoids	Nonflavanoid phenols	Proanthocyanins	Color intensity	Hue	OD280/OD315 of diluted wines	Proline
0	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065
1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050
2	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185
3	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480
4	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735
...
173	13.71	5.65	2.45	20.5	95	1.68	0.61	0.52	1.06	7.70	0.64	1.74	740
174	13.40	3.91	2.48	23.0	102	1.80	0.75	0.43	1.41	7.30	0.70	1.56	750
175	13.27	4.28	2.26	20.0	120	1.59	0.69	0.43	1.35	10.20	0.59	1.56	835
176	13.17	0.56	0.27	20.0	100	1.65	0.60	0.50	1.10	6.00	0.60	1.60	840

0-1 Normalization/Min-max normalization I

Load và hiển thị bộ dữ liệu wine dataset

```
1 import pandas as pd
2 dir = "./wine/wine.data"
3 head = ['class', 'Alcohol', 'Malic acid', 'Ash', 'Alcalinity of ash' ,
          'Magnesium', 'Total phenols', 'Flavanoids' ,
4 'Nonflavanoid phenols', 'Proanthocyanins', 'Color intensity', 'Hue',
          'OD280/OD315 of diluted wines', 'Proline']
5 df = pd.read_csv(dir, sep=",", names=head)
6 wine_data = df.iloc[:,1:]
7 wine_data
```

0-1 Normalization/Min-max normalization II

	Alcohol	Malic acid	Ash	Alcalinity of ash	Magnesium	Total phenols	Flavanoids	Nonflavanoid phenols	Proanthocyanins	Color intensity	Hue	OD280/OD315 of diluted wines	Proline	
0	14.23	1.7	2.43	15.6	127	2.80	3.06	...	0.28	2.29	5.64	1.04	3.92	1065
1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050	
2	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185	
3	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480	
4	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735	
...	
173	13.71	5.65	2.45	20.5	95	1.68	0.61	0.52	1.06	7.70	0.64	1.74	740	
174	13.40	3.91	2.48	23.0	102	1.80	0.75	0.43	1.41	7.30	0.70	1.56	750	
175	13.27	4.28	2.26	20.0	120	1.59	0.69	0.43	1.35	10.20	0.59	1.56	835	
176	13.17	2.59	2.37	20.0	120	1.65	0.68	0.53	1.46	9.30	0.60	1.62	840	
177	14.13	4.10	2.74	24.5	96	2.05	0.76	0.56	1.35	9.20	0.61	1.60	560	

178 rows x 13 columns

$\min/\max(X_j)$

0-1 Normalization/Min-max normalization III

Tính $\min(x_j)/\max(x_j)$

The minimum and maximum values of each feature column are wildly different.

```
1 wine_data.min()
```

✓ 0.0s

Python

Alcohol	11.03
Malic acid	0.74
Ash	1.36
Alcalinity of ash	10.60
Magnesium	70.00
Total phenols	0.98
Flavanoids	0.34
Nonflavanoid phenols	0.13
Proanthocyanins	0.41
Color intensity	1.28
Hue	0.48
OD280/OD315 of diluted wines	1.27
Proline	278.00
dtype: float64	

▷ ▾

```
1 wine_data.max()
```

[40]

✓ 0.0s

...	Alcohol	14.83
	Malic acid	5.80
	Ash	3.23
	Alcalinity of ash	30.00
	Magnesium	162.00
	Total phenols	3.88
	Flavanoids	5.08
	Nonflavanoid phenols	0.66
	Proanthocyanins	3.58
	Color intensity	13.00
	Hue	1.71
	OD280/OD315 of diluted wines	4.00
	Proline	1680.00
dtype: float64		

0-1 Normalization/Min-max normalization

Sau khi áp dụng 0-1 Normalization, ta thu được

```
5 df = pd.read_csv(dir, sep=";", names=head)
6 wine_data = df.iloc[:,1:]
7 wine_data
```

✓ 0.0s Python

Alcohol	Malic acid	Ash	Alcalinity of ash	Magnesium	Total phenols	Flavanoids	Nonflavanoid phenols
14.23	1.71	2.43	15.6	127	2.80	3.06	0.28
13.20	1.78	2.14	11.2	100	2.65	2.76	0.26
13.16	2.36	2.67	18.6	101	2.80	3.24	0.30
14.37	1.95	2.50	16.8	113	3.85	3.49	0.24
13.24	2.59	2.87	21.0	118	2.80	2.69	0.39
...
13.71	5.65	2.45	20.5	95	1.68	0.61	0.52
13.40	3.91	2.48	23.0	102	1.80	0.75	0.43
13.27	4.28	2.26	20.0	120	1.59	0.69	0.43
13.17	2.59	2.37	20.0	120	1.65	0.68	0.53
14.13	4.10	2.74	24.5	96	2.05	0.76	0.56

```
1 minmax_norm = (wine_data - wine_data.min()) / (wine_data.max()
- wine_data.min())
2 minmax_norm
```

[45] ✓ 0.0s Python

Alcohol	Malic acid	Ash	Alcalinity of ash	Magnesium	Total phenols	Flavanoids	Non
0.842105	0.191700	0.572193	0.257732	0.619565	0.627586	0.573840	
0.571053	0.205534	0.417112	0.030928	0.326087	0.575862	0.510549	
0.560526	0.320158	0.700535	0.412371	0.336957	0.627586	0.611814	
0.878947	0.239130	0.609626	0.319588	0.467391	0.989655	0.664557	
0.581579	0.365613	0.807487	0.536082	0.521739	0.627586	0.495781	
...	
0.705263	0.970356	0.582888	0.510309	0.271739	0.241379	0.056962	
0.623684	0.626482	0.598930	0.639175	0.347826	0.282759	0.086498	
0.589474	0.699605	0.481283	0.484536	0.543478	0.210345	0.073840	
0.563158	0.365613	0.540107	0.484536	0.543478	0.231034	0.071730	
0.815789	0.664032	0.737968	0.716495	0.282609	0.368966	0.088608	

0-1 Normalization/Min-max normalization

Rõ ràng như tên gọi 0-1 Normalization, sau khi chuẩn hoá các đặc trưng mới có giá trị nhỏ nhất là 0 và lớn nhất là 1.

```
1 minmax_norm.min()
```

✓ 0.0s Python

Alcohol	0.0
Malic acid	0.0
Ash	0.0
Alcalinity of ash	0.0
Magnesium	0.0
Total phenols	0.0
Flavanoids	0.0
Nonflavanoid phenols	0.0
Proanthocyanins	0.0
Color intensity	0.0
Hue	0.0
OD280/OD315 of diluted wines	0.0
Proline	0.0
dtype: float64	

```
1 minmax_norm.max()
```

[47] ✓ 0.0s

...	Alcohol	1.0
	Malic acid	1.0
	Ash	1.0
	Alcalinity of ash	1.0
	Magnesium	1.0
	Total phenols	1.0
	Flavanoids	1.0
	Nonflavanoid phenols	1.0
	Proanthocyanins	1.0
	Color intensity	1.0
	Hue	1.0
	OD280/OD315 of diluted wines	1.0
	Proline	1.0
dtype: float64		

Z-Score Standardization

Một chuẩn hoá tiếp theo thường hay được sử dụng hơn 0-1 Normalization là **Z-Score Standardization**

$$\text{standar}(x_j^{[i]}) = \frac{x_j^{[i]} - \text{mean}(x_j)}{\text{std}(x_j)}.$$

Trong đó, $\text{mean}(x_j)$, $\text{std}(x_j)$ lần lượt là giá trị trung bình và độ lệch chuẩn của cột đặc trưng thứ j .

After this z-score standardization, the mean of these features will be centered at *zero* and the standard deviation of the features will be *one*.

Z-Score Standardization

Sau khi áp dụng chuẩn hoá Z-Score Standardization, ta thu được các đặc trưng có giá trị như hình dưới.

```
1 znorm = (wine_data - wine_data.mean())/wine_data.std()
2 znorm
```

✓ 0.0s

Python

Alcohol	Malic acid	Ash	Alcalinity of ash	Magnesium	Total phenols	Flavanoids	Nonflavanoid phenols	Proanthocyanins	Color intensity	Hue	OD280/OD315 of diluted wines
1.514341	-0.560668	0.231400	-1.166303	1.908522	0.806722	1.031908	-0.657708	1.221438	0.251009	0.361158	1.842721
0.245597	-0.498009	-0.825667	-2.483841	0.018094	0.567048	0.731565	-0.818411	-0.543189	-0.292496	0.404908	1.110317
0.196325	0.021172	1.106214	-0.267982	0.088110	0.806722	1.212114	-0.497005	2.129959	0.268263	0.317409	0.786369
1.686791	-0.345835	0.486554	-0.806975	0.928300	2.484437	1.462399	-0.979113	1.029251	1.182732	-0.426341	1.180741
0.294868	0.227053	1.835226	0.450674	1.278379	0.806722	0.661485	0.226158	0.400275	-0.318377	0.361158	0.448336
...
0.873810	2.966176	0.304301	0.300954	-0.331985	-0.982841	-1.420891	1.270726	-0.927563	1.139596	-1.388840	-1.227742
0.491955	1.408636	0.413653	1.049555	0.158126	-0.791103	-1.280731	0.547563	-0.316058	0.967055	-1.126341	-1.481267
0.331822	1.739837	-0.388260	0.151234	1.418411	-1.126646	-1.340800	0.547563	-0.420888	2.217979	-1.607590	-1.481267
0.208643	0.227053	0.012696	0.151234	1.418411	-1.030776	-1.350811	1.351077	-0.228701	1.829761	-1.563840	-1.396759
1.391162	1.578712	1.361368	1.498716	-0.261969	-0.391646	-1.270720	1.592131	-0.420888	1.786626	-1.520090	-1.424928

Z-Score Standardization

Zero Mean & Standard Deviation 1

Sau khi chuẩn hoá Z-Score Standardization, kiểm tra xem **trung bình** của các cột đặc trưng bằng **0** và **độ lệch chuẩn** bằng **1**.

```
1 znorm.mean()
```

✓ 0.0s

Python

Alcohol	-9.181170e-16
Malic acid	0.000000e+00
Ash	-8.070947e-16
Alcalinity of ash	-7.983626e-17
Magnesium	-1.995907e-17
Total phenols	3.991813e-17
Flavanoids	-3.592632e-16
Nonflavanoid phenols	3.592632e-16
Proanthocyanins	-1.596725e-16
Color intensity	1.995907e-17
Hue	1.995907e-16
OD280/OD315 of diluted wines	3.193450e-16
Proline	-7.983626e-17

dtype: float64

```
1 znorm.std()
```

[56]

✓ 0.0s

...	Alcohol	1.0
	Malic acid	1.0
	Ash	1.0
	Alcalinity of ash	1.0
	Magnesium	1.0
	Total phenols	1.0
	Flavanoids	1.0
	Nonflavanoid phenols	1.0
	Proanthocyanins	1.0
	Color intensity	1.0
	Hue	1.0
	OD280/OD315 of diluted wines	1.0
	Proline	1.0

dtype: float64

Lưu ý, khi chuẩn hoá dữ liệu đầu vào thì phải chuẩn hoá cho cả 3 tập: training, validation và test set.

Chuẩn hoá cho 3 tập sử dụng 0-1 Normalization

```
1 def normalize(df, train_min, train_max):
2     return (df - train_min) / (train_max - train_min)
3
4 train_min, train_max = df_train.min(), df_train.max()
5 df_train_norm = normalize(df_train, train_min, train_max)
6 df_val_norm = normalize(df_val, train_min, train_max)
7 df_test_norm = normalize(df_test, train_min, train_max)
```

Thực hiện tương tự cho Z-score standardization trên 3 tập.

Tài liệu tham khảo



Sebastian Raschka, Yuxi (Hayden) Liu, Vahid Mirjalili
Machine Learning with PyTorch and Scikit-Learn: Develop
machine learning and deep learning models with Python
(2022). Published by Packt Publishing Ltd, ISBN
978-1-80181-931-2.



Sebastian Raschka
MACHINE LEARNING Q AND AI: 30 Essential Questions
and Answers on Machine Learning and AI (2024). ISBN-13:
978-1-7185-0377-9 (ebook).



LightningAI
LightningAI: PyTorch Lightning (2024) .