AI VIET NAM – AI COURSE 2024

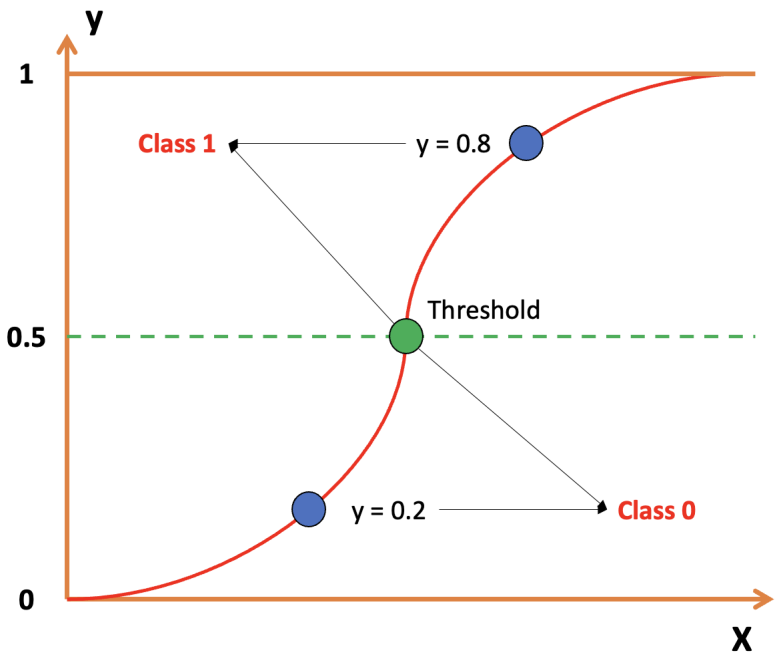
Exercise: Logistic Regression

Dinh-Thang Duong, Quang-Vinh Dinh

Ngày 26 tháng 10 năm 2024

Phần I: Giới thiệu

Logistic Regression là một trong những thuật toán supervised-learning Machine Learning nền tảng quan trong nhất, được sử dụng để giải quyết bài toán Phân loại nhị phân (Binary Classification). Logistic Regression phân tích mối quan hệ giữa các biến phụ thuộc và biến độc lập nhị phân trong dữ liệu huấn luyện, từ đó có thể ước lượng xác suất phân lớp cho một mẫu dữ liệu mới.



Hình 1: Minh họa về Logistic Regression.

Trong bài tập này ở phần lập trình, chúng ta sẽ thực hành cài đặt từ đầu quá trình xây dựng một mô hình Logistic Regression, áp dụng vào giải quyết hai bài toán phân loại nhị phân là Titanic Survival Prediction và Twitter Sentiment Analysis. Đồng thời, ôn tập một số lý thuyết về Logistic Regression thông qua bài tập trắc nghiệm.

Phần II: Bài tập

A. Phần lập trình

- **Titanic Survival Prediction**

1. **Tải bộ dữ liệu:** Các bạn tải bộ dữ liệu tại đây.
2. **Import libraries:**

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

3. **Đọc dữ liệu:** Sử dụng thư viện pandas để đọc file .csv thành DataFrame như sau:

```python
dataset_path = 'titanic_modified_dataset.csv'
df = pd.read_csv(
    dataset_path,
    index_col='PassengerId'
)
```

PassengerId	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Title	Survived
1	3	0	22.0	1	0	7.2500	0	0	0
2	1	1	38.0	1	0	71.2833	1	1	1
3	3	1	26.0	0	0	7.9250	0	2	1
4	1	1	35.0	1	0	53.1000	0	1	1
5	3	0	35.0	0	0	8.0500	0	0	0
...	...	...	...	...	...	...	...	...	...
887	2	0	27.0	0	0	13.0000	0	5	0
888	1	1	19.0	0	0	30.0000	0	2	1
889	3	1	28.0	1	2	23.4500	0	2	0
890	1	0	26.0	0	0	30.0000	1	0	1
891	3	0	32.0	0	0	7.7500	2	0	0

891 rows × 9 columns

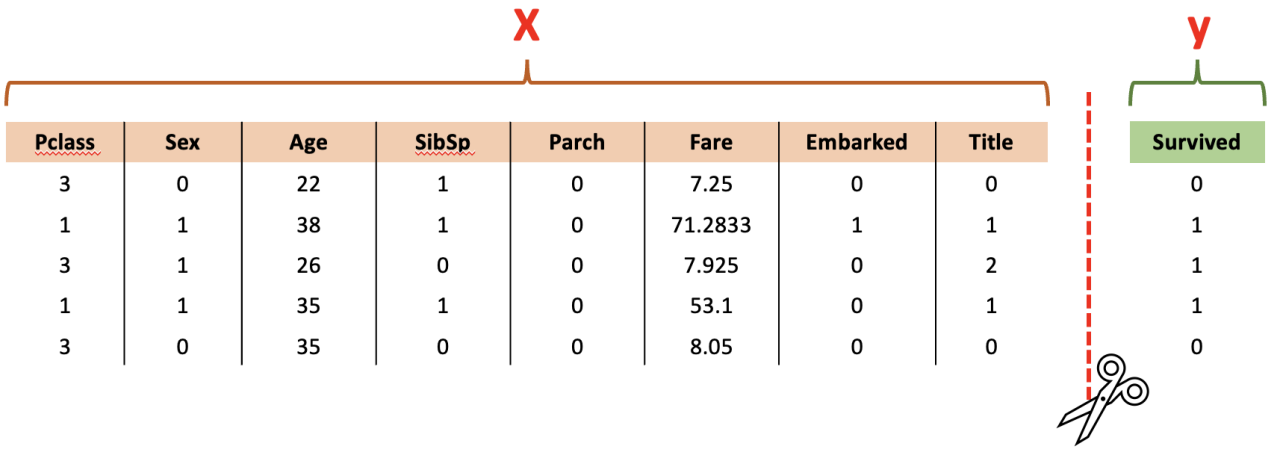
Hình 2: DataFrame của bộ dữ liệu Titanic Survival Prediction.

Trong đó:

- **PassengerId:** Mã hàng khách. Đây được xem là **chỉ mục** của bảng dữ liệu.
- **Pclass:** Hạng vé tàu của hàng khách.
- **Sex:** Giới tính của hàng khách.
- **Age:** Tuổi của hàng khách.
- **SibSp:** Số lượng anh chị em và/hoặc người yêu đi cùng chuyến tàu với hàng khách.
- **Parch:** Số lượng phụ huynh và/hoặc con cháu đi cùng chuyến tàu với hàng khách.
- **Fare:** Giá vé tàu của hàng khách.
- **Embarked:** Cảng xuất phát của hàng khách.
- **Title:** Tước hiệu của hàng khách.
- **Survived:** Hàng khách có (1) sống sót qua thảm kịch hay không (0)?

4. **Chia biến X, y:** Chuyển đổi DataFrame hiện tại thành array và tách hai biến X, y:

```
dataset_arr = df.to_numpy().astype(np.float64)
X, y = dataset_arr[:, :-1], dataset_arr[:, -1]
```



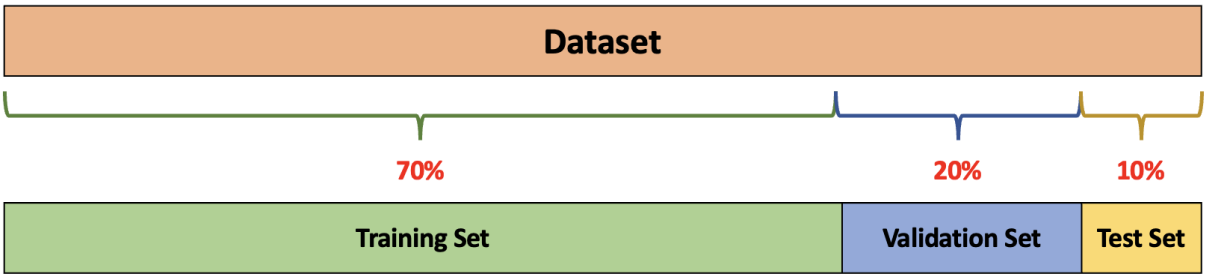
Hình 3: Mô phỏng việc tách biến X và y từ bộ dữ liệu gốc.

5. **Thêm bias vào X:** Khi sử dụng thư viện, bias sẽ được thêm tự động vào X. Tuy nhiên, khi triển khai lại từ đầu, chúng ta cần phải tự thêm bias vào mỗi mẫu dữ liệu, nhằm thỏa mãn công thức hàm dự đoán:

```
intercept = np.ones((
    X.shape[0], 1)
)
X_b = np.concatenate(
    (intercept, X),
    axis=1
)
```

6. **Chia tập train, val, test:** Sau khi đã hoàn chỉnh biến X, chúng ta tiến hành chia ba bộ train, val, test với tỉ lệ 7:2:1. Thực hiện như sau:

```
val_size = 0.2
test_size = 0.125
random_state = 2
is_shuffle = True

X_train, X_val, y_train, y_val = train_test_split(
    X_b, y,
    test_size=val_size,
    random_state=random_state,
    shuffle=is_shuffle
)

X_train, X_test, y_train, y_test = train_test_split(
    X_train, y_train,
    test_size=test_size,
    random_state=random_state,
    shuffle=is_shuffle
)
```



Hình 4: Mô phỏng chia bộ dữ liệu gốc thành ba bộ train, val, test với tỉ lệ 7:2:1.

7. **Chuẩn hóa dữ liệu:** Ta sử dụng X_train vừa tạo ở bước trên fit vào hàm chuẩn hóa StandardScaler. Sau đó, đem scaler này chuẩn hóa cho tập X_val và X_test (lưu ý rằng ta không chuẩn hóa bias nên sẽ bỏ qua cột đầu tiên trong X):

```
normalizer = StandardScaler()
X_train[:, 1:] = normalizer.fit_transform(X_train[:, 1:])
X_val[:, 1:] = normalizer.transform(X_val[:, 1:])
X_test[:, 1:] = normalizer.transform(X_test[:, 1:])
```

8. **Cài đặt các hàm quan trọng:** Để thuận tiện trong việc cài đặt chương trình, ta định nghĩa sẵn một số hàm sẽ được dùng trong quá trình huấn luyện mô hình:
   - **Hàm sigmoid:** Xây dựng hàm sigmoid với công thức như sau:

$$sigmoid(Z) = \frac{1}{1 + e^{-Z}}$$

```
def sigmoid(z):
    return 1 / (1 + np.exp(-z))
```

   - **Hàm dự đoán:**

```python
def predict(X, theta):
    dot_product = np.dot(X, theta)
    y_hat = sigmoid(dot_product)

    return y_hat
```

– **Hàm tính loss:** Xây dựng hàm tính loss với công thức Cross-entropy như sau:

$$loss(y, y\_hat) = -\frac{1}{batch\_size} \sum_{i=1}^{batch\_size} (y_i \times \log(y\_hat_i) + (1-y_i) \times \log(1-y\_hat_i))$$

```python
def compute_loss(y_hat, y):
    y_hat = np.clip(
        y_hat, 1e-7, 1 - 1e-7
    )

    return (
        -y * \
        np.log(y_hat) - (1 - y) * \
        np.log(1 - y_hat)
    ).mean()
```

– **Hàm tính gradient:** Xây dựng hàm tính gradient với công thức như sau:

$$gradient(X, y, y\_hat) = \frac{X^T \cdot (y\_hat - y)}{batch\_size}$$

```python
def compute_gradient(X, y, y_hat):
    return np.dot(
        X.T, (y_hat - y)
    ) / y.size
```

– **Hàm cập nhật trọng số:** Khi áp dụng giải thuật Gradient Descent, trọng số *theta* sẽ được cập nhật bằng công thức như sau:

$$theta = theta - learning\_rate \times gradient$$

```python
def update_theta(theta, gradient, lr):
    return theta - lr * gradient
```

– **Hàm tính độ chính xác:** Xây dựng hàm tính độ chính xác với công thức như sau:

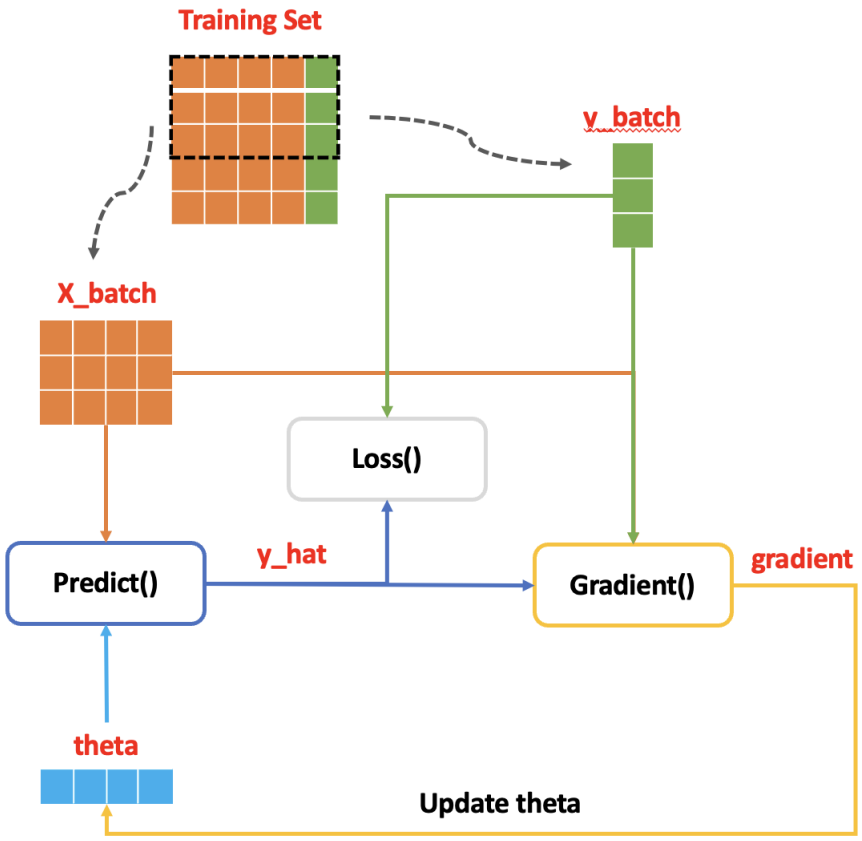
$$accuracy = \frac{\text{Số lần dự đoán đúng}}{\text{Tổng số lần dự đoán}}$$

```python
def compute_accuracy(X, y, theta):
    y_hat = predict(X, theta).round()
    acc = (y_hat == y).mean()

    return acc
```

9. **Khai báo các siêu tham số và khởi tạo weights:**

```
lr = 0.01
epochs = 100
batch_size = 16

np.random.seed(random_state)
theta = np.random.uniform(
    size=X_train.shape[1]
)
```

10. **Huấn luyện mô hình:** Chúng ta sẽ triển khai quá trình huấn luyện mô hình với ý tưởng chính như sau: Khởi tạo vòng lặp với số lần lặp bằng số **epochs**. Với mỗi lần lặp, duyệt qua toàn bộ mẫu dữ liệu (trong training set) theo từng bộ mẫu dữ liệu có kích thước **batch_size** (tạm gọi là cặp **X_i** và **y_i**) và thực hiện các bước tính toán sau:
    (a) Tính **y_hat** sử dụng hàm **predict(X_i, theta)**. Đây là kết quả dự đoán của mô hình với các mẫu dữ liệu tại batch đang xét.
    (b) Tính **loss** sử dụng hàm **compute_loss(y_hat, y_i)**. Lưu trữ giá trị này vào một list **batch_losses**, dùng cho việc trực quan hóa kết quả huấn luyện sau này.
    (c) Tính **gradient** sử dụng hàm **compute_gradient(X_i, y_i, y_hat)**.
    (d) Sử dụng kết quả gradient vừa tìm được để cập nhật bộ trọng số **theta** sử dụng hàm **update_theta(theta, gradient, lr)**.



Hình 5: Mô tả quá trình huấn luyện mô hình Logistic Regression sử dụng Gradient Descent.

Tổng kết lại, chúng ta sẽ có toàn bộ code cài đặt như sau:

```
train_accs = []
train_losses = []
val_accs = []
val_losses = []

for epoch in range(epochs):
    train_batch_losses = []
    train_batch_accs = []
    val_batch_losses = []
    val_batch_accs = []

    for i in range(0, X_train.shape[0], batch_size):
        X_i = X_train[i:i+batch_size]
        y_i = y_train[i:i+batch_size]

        y_hat = predict(X_i, theta)

        train_loss = compute_loss(y_hat, y_i)

        gradient = compute_gradient(X_i, y_i, y_hat)

        theta = update_theta(theta, gradient, lr)


        train_batch_losses.append(train_loss)

        train_acc = compute_accuracy(X_train, y_train, theta)
        train_batch_accs.append(train_acc)

        y_val_hat = predict(X_val, theta)
        val_loss = compute_loss(y_val_hat, y_val)
        val_batch_losses.append(val_loss)

        val_acc = compute_accuracy(X_val, y_val, theta)
        val_batch_accs.append(val_acc)

    train_batch_loss = sum(train_batch_losses) / len(
train_batch_losses)
    val_batch_loss = sum(val_batch_losses) / len(val_batch_losses)
    train_batch_acc = sum(train_batch_accs) / len(train_batch_accs)
    val_batch_acc = sum(val_batch_accs) / len(val_batch_accs)

    train_losses.append(train_batch_loss)
    val_losses.append(val_batch_loss)
    train_accs.append(train_batch_acc)
    val_accs.append(val_batch_acc)

    print(f'\nEPOCH {epoch + 1}:\tTraining loss: {train_batch_loss:.3
f}\tValidation loss: {val_batch_loss:.3f}')
```

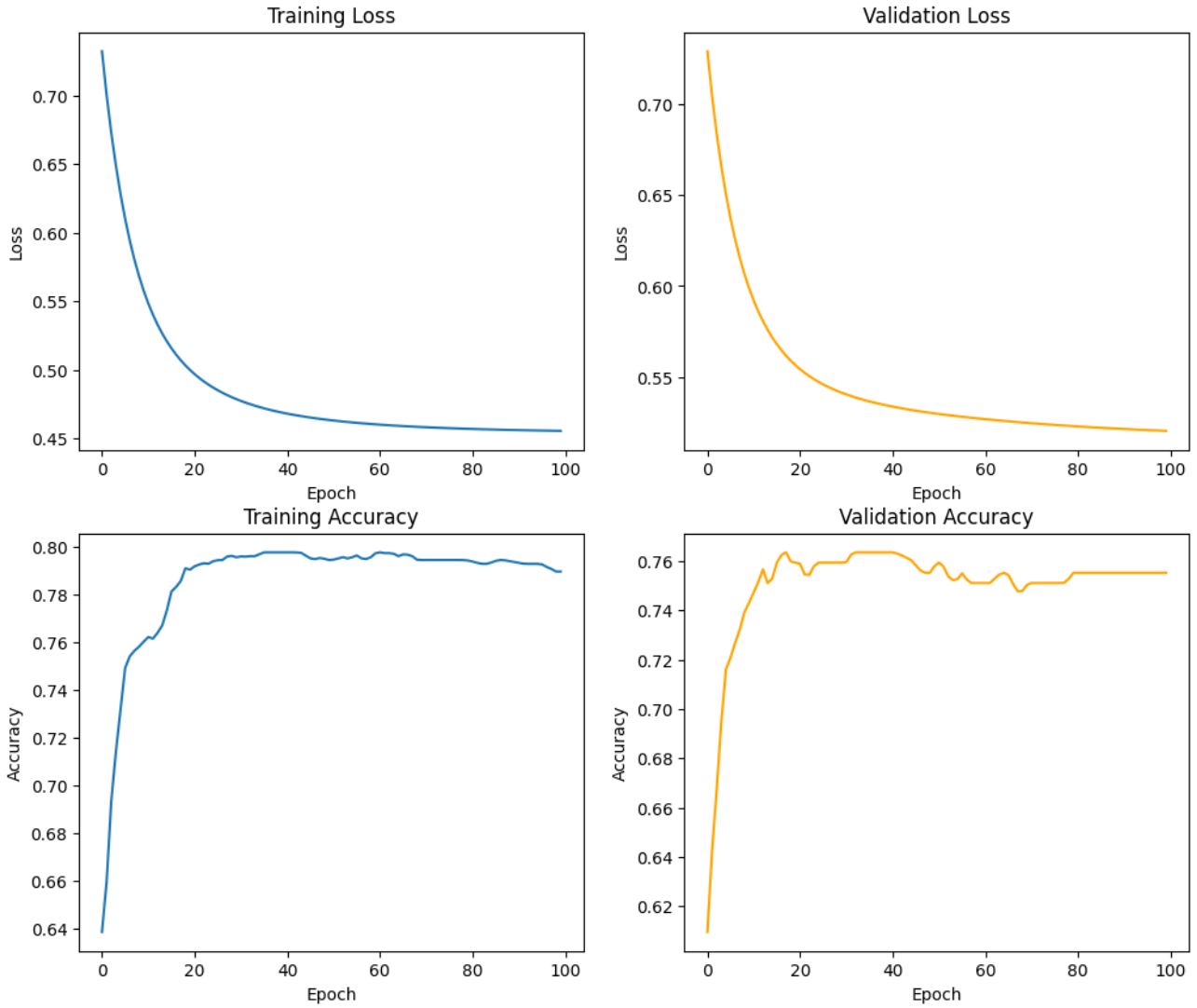
Khi chạy thuật toán, nếu các bạn quan sát thấy giá trị loss giảm và độ chính xác tăng dần khi số epoch tăng, điều đó là dấu hiệu cho thấy code huấn luyện mô hình của chúng ta hoạt động ổn.

```
EPOCH 90:       Training loss: 0.456    Validation loss: 0.522
EPOCH 91:       Training loss: 0.456    Validation loss: 0.521
EPOCH 92:       Training loss: 0.456    Validation loss: 0.521
EPOCH 93:       Training loss: 0.456    Validation loss: 0.521
EPOCH 94:       Training loss: 0.456    Validation loss: 0.521
EPOCH 95:       Training loss: 0.456    Validation loss: 0.521
EPOCH 96:       Training loss: 0.456    Validation loss: 0.521
EPOCH 97:       Training loss: 0.456    Validation loss: 0.521
EPOCH 98:       Training loss: 0.456    Validation loss: 0.521
EPOCH 99:       Training loss: 0.456    Validation loss: 0.521
EPOCH 100:      Training loss: 0.456    Validation loss: 0.520
```

Hình 6: Kết quả huấn luyện in trên màn hình ở những epoch cuối cùng

Bên cạnh đó, với các danh sách batch loss và batch accuracy trên hai bộ dữ liệu train và val, chúng ta còn có thể trực quan hóa kết quả huấn luyện lên đồ thị như sau:

```python
fig, ax = plt.subplots(2, 2, figsize=(12, 10))
ax[0, 0].plot(train_losses)
ax[0, 0].set(xlabel='Epoch', ylabel='Loss')
ax[0, 0].set_title('Training Loss')

ax[0, 1].plot(val_losses, 'orange')
ax[0, 1].set(xlabel='Epoch', ylabel='Loss')
ax[0, 1].set_title('Validation Loss')

ax[1, 0].plot(train_accs)
ax[1, 0].set(xlabel='Epoch', ylabel='Accuracy')
ax[1, 0].set_title('Training Accuracy')

ax[1, 1].plot(val_accs, 'orange')
ax[1, 1].set(xlabel='Epoch', ylabel='Accuracy')
ax[1, 1].set_title('Validation Accuracy')

plt.show()
```



Hình 7: Hình ảnh trực quan kết quả huấn luyện trên tập train và val cho bài Titanic Survival Prediction.

11. **Đánh giá mô hình:** Sử dụng bộ trọng số mô hình tìm được sau quá trình huấn luyện, ta đánh giá độ chính xác của mô hình trên hai tập val và test:

```
val_set_acc = compute_accuracy(X_val, y_val, theta)
test_set_acc = compute_accuracy(X_test, y_test, theta)
print('Evaluation on validation and test set:')
print(f'Accuracy: {val_set_acc}')
print(f'Accuracy: {test_set_acc}')
```

- **Twitter Sentiment Analysis**

1. **Tải bộ dữ liệu:** Các bạn tải bộ dữ liệu tại đây.
2. **Import libraries:**

```
import pandas as pd
import numpy as np
```

```
import re
import nltk
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from nltk.tokenize import TweetTokenizer
from collections import defaultdict
```

3. **Đọc bộ dữ liệu:** Sử dụng thư viện pandas để đọc file .csv thành DataFrame:

```
dataset_path = 'sentiment_analysis.csv'
df = pd.read_csv(
    dataset_path,
    index_col='id'
)
```

id	label	tweet
1	0	#fingerprint #Pregnancy Test https://goo.gl/h1...
2	0	Finally a transparant silicon case ^^ Thanks t...
3	0	We love this! Would you go? #talk #makememorie...
4	0	I'm wired I know I'm George I was made that wa...
5	1	What amazing service! Apple won't even talk to...
...	...	...
7916	0	Live out loud #lol #liveoutloud #selfie #smile...
7917	0	We would like to wish you an amazing day! Make...
7918	0	Helping my lovely 90 year old neighbor with he...
7919	0	Finally got my #smart #pocket #wifi stay conne...
7920	0	Apple Barcelona!!! #Apple #Store #BCN #Barcelo...

7920 rows × 2 columns

Hình 8: DataFrame của bộ dữ liệu Twitter Sentiment Analysis.

4. **Tiền xử lý bộ dữ liệu:** Dữ liệu đầu vào của chúng ta lúc này hiện đang ở dạng văn bản (string), chưa có đặc trưng rõ ràng cũng như không thể đưa vào huấn luyện mô hình được. Vì vậy, chúng ta sẽ tiền xử lý dữ liệu văn bản đầu vào để đưa về một dạng vector đặc trưng nào đó:

   (a) **Xây dựng hàm chuẩn hóa văn bản:** Văn bản gốc có rất nhiều kí tự thừa thải, vô nghĩa... Vì vậy, ta cần loại bỏ chúng cũng như áp dụng thêm vài các bước chuẩn hóa văn bản khác để văn bản đầu vào trở nên ít phức tạp hơn, nhằm tăng cường hiệu quả biểu diễn của vector đặc trưng sau này:

```python
def text_normalize(text):
    # Retweet old acronym "RT" removal
    text = re.sub(r'^RT[\s]+', '', text)

    # Hyperlinks removal
    text = re.sub(r'https?:\/\/.*[\r\n]*', '', text)

    # Hashtags removal
    text = re.sub(r'#', '', text)

     # Punctuation removal
    text = re.sub(r'[^\w\s]', '', text)

    # Tokenization
    tokenizer = TweetTokenizer(
        preserve_case=False,
        strip_handles=True,
        reduce_len=True
    )
    text_tokens = tokenizer.tokenize(text)

    return text_tokens
```

Trong đó:

- **Dòng 1:** Khai báo hàm text_normalize() nhận đầu vào là một string (text).
- Dòng 2, 3: Loại bỏ các từ "RT"trong text (đây là một cụm từ viết tắt cũ cho "Retweet").
- **Dòng 5, 6:** Loại bỏ các đường dẫn trong text.
- **Dòng 8, 9:** Loại bỏ các hashtag.
- **Dòng 11, 12:** Loại bỏ các dấu câu.
- **Dòng 14, 15, 16, 17, 18, 19:** Khai báo tokenizer.
- **Dòng 20:** Tokenize text (kết quả trả về là danh sách các token).
- **Dòng 22:** Trả về danh sách các token.

(b) **Xây dựng bộ lưu giữ tần suất xuất hiện của các từ:** Có rất nhiều cách để ta có thể tạo vector biểu diễn cho một đoạn văn bản. Trong bài tập này, chúng ta sẽ sử dụng loại vector lưu trữ số lần xuất hiện của các từ thuộc class "positive"và các từ thuộc class "negative"trong một văn bản. Để làm được điều này, đầu tiên chúng ta cần phải xây dựng một bộ từ điển lưu trữ tần suất xuất hiện của toàn bộ mọi từ trong bộ dữ liệu với class tương ứng của nó. Cách làm như sau:

```python
def get_freqs(df):
    freqs = defaultdict(lambda: 0)
    for idx, row in df.iterrows():
        tweet = row['tweet']
        label = row['label']

        tokens = text_normalize(tweet)
        for token in tokens:
            pair = (token, label)
            freqs[pair] += 1

    return freqs
```

Trong đó:

- **Dòng 1:** Khai báo hàm get_freqs() với tham số đầu vào là DataFrame chứa bộ dữ liệu (df).
- **Dòng 2:** Khai báo một defaultdict (defaultdict khác với dict thông thường ở điểm defaultdict tự động gán giá trị mặc định cho các key mới, ở đây ta gán bằng 0).
- **Dòng 3, 4, 5:** Duyệt qua từng dòng tweet và label tương ứng:
- **Dòng 7:** Chuẩn hóa dòng tweet hiện tại.
- **Dòng 8, 9, 10:** Duyệt qua từng từ (token) trong tweet hiện tại, khai báo key có dạng tuple (token, label) và tăng giá trị của key lên 1.
- **Dòng 12:** Trả về dictionary lưu giữ tần suất xuất hiện của các từ.

(c) **Xây dựng hàm tạo vector đặc trưng:** Kết hợp hai thành phần trên, ta xây dựng một hàm tạo vector đặc trưng cho văn bản đầu vào. Cách làm như sau:

```
1  def get_feature(text, freqs):
2      tokens = text_normalize(text)
3
4      X = np.zeros(3)
5      X[0] = 1
6
7      for token in tokens:
8          X[1] += freqs[(token, 0)]
9          X[2] += freqs[(token, 1)]
10
11     return X
```

Trong đó:

- **Dòng 1:** Khai báo hàm get_feature() nhận tham số đầu vào là đoạn văn bản (text) và dictionary lưu giữ tần suất xuất hiện các từ (freqs).
- **Dòng 2:** Chuẩn hóa văn bản đầu vào.
- **Dòng 4:** Tạo một vector biểu diễn văn bản giá trị 0 có 3 phần tử, đại diện cho (intercept, n_positives, n_negatives)
- **Dòng 5:** Gán phần tử đầu tiên giá trị 1 (intercept).
- **Dòng 7, 8, 9:** Duyệt qua từng từ trong văn bản đầu vào, lấy giá trị tần suất của từ ứng với từng label và cộng dồn vào vị trí phần tử trong vector biểu diễn tương ứng.
- **Dòng 11:** Trả về vector biểu diễn.

(d) **Trích xuất đặc trưng toàn bộ dữ liệu:** Cuối cùng, ta sử dụng hàm get_feature() ở trên để đổi toàn bộ văn bản thành vector biểu diễn mới như sau:

```
1  X = []
2  y = []
3
4  freqs = get_freqs(df)
5  for idx, row in df.iterrows():
6      tweet = row['tweet']
7      label = row['label']
8
9      X_i = get_feature(tweet, freqs)
```

```
        X.append(X_i)
        y.append(label)

X = np.array(X)
y = np.array(y)
```

5. **Chia bộ train, val, test:** Thực hiện tương tự như bài Titanic.

```
val_size = 0.2
test_size = 0.125
random_state = 2
is_shuffle = True

X_train, X_val, y_train, y_val = train_test_split(
    X, y,
    test_size=val_size,
    random_state=random_state,
    shuffle=is_shuffle
)

X_train, X_test, y_train, y_test = train_test_split(
    X_train, y_train,
    test_size=test_size,
    random_state=random_state,
    shuffle=is_shuffle
)
```

6. **Chuẩn hóa dữ liệu:** Thực hiện tương tự như bài Titanic.

```
normalizer = StandardScaler()
X_train[:, 1:] = normalizer.fit_transform(X_train[:, 1:])
X_val[:, 1:] = normalizer.transform(X_val[:, 1:])
X_test[:, 1:] = normalizer.transform(X_test[:, 1:])
```

7. **Cài đặt các hàm quan trọng:** Sử dụng lại các hàm đã định nghĩa trong bài Titanic.

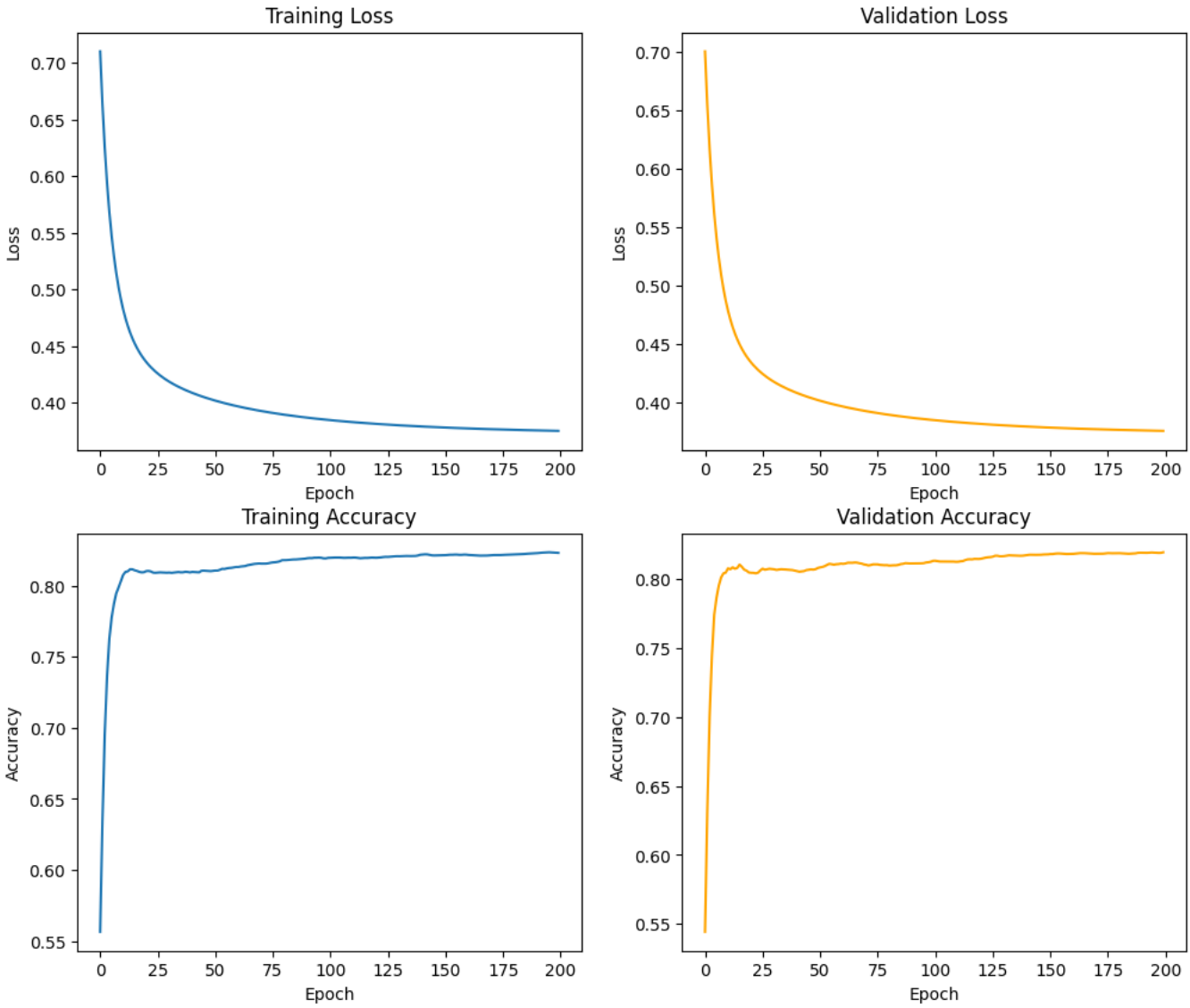
```
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

def compute_loss(y_hat, y):
    y_hat = np.clip(
        y_hat, 1e-7, 1 - 1e-7
    )

    return (-y * np.log(y_hat) - (1 - y) * np.log(1 - y_hat)).mean()

def predict(X, theta):
    dot_product = np.dot(X, theta)
    y_hat = sigmoid(dot_product)

    return y_hat

def compute_gradient(X, y, y_hat):
    return np.dot(
        X.T, (y_hat - y)
    ) / y.size

```

```python
def update_theta(theta, gradient, lr):
    return theta - lr * gradient

def compute_accuracy(X, y, theta):
    y_hat = predict(X, theta).round()
    acc = (y_hat == y).mean()

    return acc
```

8. **Khai báo các siêu tham số và khởi tạo weights:** Trong bài này, vì số lượng mẫu dữ liệu nhiều hơn bài Titanic, ta có thể cân nhắc tăng số batch size lên để tăng tốc độ huấn luyện (ví dụ ở đây ta cài batch_size=128).

```python
lr = 0.01
epochs = 200
batch_size = 128

np.random.seed(random_state)
theta = np.random.uniform(
    size=X_train.shape[1]
)
```

9. **Huấn luyện mô hình:** Sử dụng code huấn luyện tương tự như trong bài Titanic. Kết quả của quá trình huấn luyện được trực quan trên đồ thị như sau:



Hình 9: Hình ảnh trực quan kết quả huấn luyện trên tập train và val cho bài Twitter Sentiment Analysis.

10. **Đánh giá mô hình:** Sử dụng code đánh giá tương tự như trong bài Titanic:

```
val_set_acc = compute_accuracy(X_val, y_val, theta)
test_set_acc = compute_accuracy(X_test, y_test, theta)
print('Evaluation on validation and test set:')
print(f'Accuracy: {val_set_acc}')
print(f'Accuracy: {test_set_acc}')
```

B. Phần trắc nghiệm

1. Logistic Regression thuộc nhánh nào trong Machine Learning?
   (*a*) Supervised Learning.
   (*b*) Unsupervised Learning.
   (*c*) Reinforcement Learning.
   (*d*) Self-supervised Learning.

2. Logistic Regression thường được áp dụng để giải quyết vấn đề nào dưới đây?
   (*a*) Phân lớp nhị phân.
   (*b*) Dự đoán chuỗi thời gian.
   (*c*) Giảm chiều dữ liệu.
   (*d*) Phân cụm dữ liệu.

3. Bài toán nào sau đây có thể được giải quyết hiệu quả bằng Logistic Regression?
   (*a*) Dự đoán giá nhà.
   (*b*) Phân loại email spam.
   (*c*) Đề xuất phim.
   (*d*) Dự đoán giá cổ phiếu.

4. Hàm Cross-Entropy được chọn làm hàm loss trong Logistic Regression vì lý do gì?
   (*a*) Nó giúp quá trình huấn luyện nhanh hơn.
   (*b*) Nó giữ giá trị dự đoán trong khoảng $[0, 1]$.
   (*c*) Nó có dạng lồi, dễ tối ưu.
   (*d*) Nó giảm lỗi dự đoán một cách hiệu quả.

5. Trong Logistic Regression, nếu giá trị batch_size được cài đặt bằng 1, kiểu huấn luyện này được gọi là gì?
   (*a*) Batch Gradient Descent.
   (*b*) Stochastic Gradient Descent.
   (*c*) Mini-batch Gradient Descent.
   (*d*) Standard Gradient Descent.

6. Với một mẫu dữ liệu được phân loại chính xác bởi Logistic Regression, giá trị loss của mẫu này sẽ như thế nào?
   (*a*) Bằng 0.5.
   (*b*) Gần bằng 0.
   (*c*) Bằng 1.
   (*d*) Gần bằng 0.5.

7. Hàm nào sau đây mô tả đúng gradient trong quá trình tối ưu Logistic Regression?
   (a) $\nabla J(\theta) = \frac{1}{m} X^T (h_\theta(X) - y)$
   (b) $\nabla J(\theta) = \frac{1}{m} X (y - h_\theta(X))$
   (c) $\nabla J(\theta) = \frac{1}{m} X (h_\theta(X) - y)$
   (d) $\nabla J(\theta) = \frac{1}{m} \sum (h_\theta(X) - y)$

8. Hàm Sigmoid trả về giá trị trong khoảng nào?
   (a) $[-1, 1]$
   (b) $(0, 1)$
   (c) $(0, \infty)$
   (d) $[-\infty, 0]$

9. Trong quá trình huấn luyện mô hình Logistic Regression sử dụng Gradient Descent, khi cài đặt batch_size nhỏ hơn số lượng mẫu ($1 <$ batch_size $<$ n_samples), kỹ thuật này gọi là gì?
   (a) Stochastic Gradient Descent.
   (b) Mini-batch Gradient Descent.
   (c) Batch Gradient Descent.
   (d) Standard Gradient Descent.

10. Đâu là lý do chính khi Logistic Regression không sử dụng Mean Squared Error làm hàm loss?
    (a) Vì Cross-Entropy dễ tối ưu hơn cho phân loại nhị phân.
    (b) Vì Mean Squared Error không hội tụ.
    (c) Vì Mean Squared Error chỉ phù hợp với hồi quy tuyến tính.
    (d) Mean Squared Error làm mô hình dễ bị overfitting.

11. Hàm nào sau đây mô tả đúng hàm loss trong Logistic Regression với y là giá trị thực và $h_\theta(x)$ là giá trị dự đoán?
    (a) $L(y, h_\theta(x)) = -[y \log(h_\theta(x)) + (1 - y) \log(1 - h_\theta(x))]$
    (b) $L(y, h_\theta(x)) = (y - h_\theta(x))^2$
    (c) $L(y, h_\theta(x)) = |y - h_\theta(x)|$
    (d) $L(y, h_\theta(x)) = y \log(1 - h_\theta(x)) + (1 - y) \log(h_\theta(x))$

12. Trong các độ đo dưới đây, độ đo nào thường không được sử dụng để đánh giá một mô hình Logistic Regression?
    (a) Accuracy.
    (b) Precision.
    (c) Binary Cross Entropy.
    (d) Mean Absolute Error.

13. Cho đoạn chương trình sau:

```
def predict(X, theta):
    z = np.dot(X, theta)

    return 1 / (1 + np.exp(-z))
```

Khi truyền vector **X** = [[22.3, -1.5, 1.1, 1]] và vector **theta** = [0.1, -0.15, 0.3, -0.2] vào hàm `predict()` trên, kết quả trả về của hàm là:

(*a*) 0.14239088

(*b*) 0.71259201

(*c*) 0.92988994

(*d*) 0.54991232

14. Cho đoạn chương trình sau:

```
def compute_loss(y_hat, y):
    y_hat = np.clip(
        y_hat, 1e-7, 1 - 1e-7
    )

    return (-y * np.log(y_hat) - (1 - y) * np.log(1 - y_hat)).mean()
```

Khi truyền vector **y** = np.array([1, 0, 0, 1]) và vector **y_hat** = np.array([0.8, 0.75, 0.3, 0.95]) vào hàm `compute_loss()` trên, kết quả trả về của hàm là (làm tròn đến hàng thập phân thứ 3):

(*a*) 0.504

(*b*) 0.201

(*c*) 0.921

(*d*) 0.623

15. Khi mô hình Logistic Regression dự đoán giá trị 0.8 trong bài toán phân loại cảm xúc, điều đó có nghĩa là gì?

(*a*) Văn bản có 80% tỉ lệ là tiêu cực.

(*b*) Văn bản có 80% tỉ lệ là tích cực.

(*c*) Văn bản có 20% tỉ lệ là tích cực.

(*d*) Không xác định được tỉ lệ.

16. Cho đoạn chương trình sau:

```
def compute_gradient(X, y_true, y_pred):
    gradient = np.dot(X.T, (y_pred - y_true)) / y_true.size

    return gradient

```

Khi truyền **X** = [[1, 2], [2, 1], [1, 1], [2, 2]], **y_true** = [0, 1, 0, 1] và **y_pred** = [0.25, 0.75, 0.4, 0.8] vào hàm `compute_gradient()` trên, kết quả trả về của hàm là:

(a) $[0.100, 0.250]$

(b) $[0.150, 0.200]$

(c) $[0.125, 0.225]$

(d) $[0.175, 0.275]$

17. Cho đoạn chương trình sau:

```
def compute_accuracy(y_true, y_pred):
    y_pred_rounded = np.round(y_pred)
    accuracy = np.mean(y_true == y_pred_rounded)

    return accuracy

```

Khi truyền vector **y_true** = [1, 0, 1, 1] và **y_pred** = [0.85, 0.35, 0.9, 0.75] vào hàm `compute_accuracy()` trên, kết quả trả về của hàm là:

(a) 0.75

(b) 0.80

(c) 0.90

(d) 1.00

18. Cho đoạn chương trình sau:

```
def compute_gradient(X, y_true, y_pred):
    gradient = np.dot(X.T, (y_pred - y_true)) / y_true.size

    return gradient

```

Khi truyền **X** = [[1, 3], [2, 1], [3, 2], [1, 2]], **y_true** = [1, 0, 1, 1] và **y_pred** = [0.7, 0.4, 0.6, 0.85] vào hàm `compute_gradient()` trên, kết quả trả về của hàm là:

(a) $[0.0375, 0.175]$

(b) $[0.025, 0.15]$

(c) $[0.045, 0.20]$

(d) $[0.05, 0.25]$

- Hết -