

ĐẠI HỌC CÔNG NGHỆ THÔNG TIN



UIT

BÁO CÁO TIỂU LUẬN

MÔN: THỊ GIÁC MÁY TÍNH NÂNG CAO

SOFTMAX REGRESSION

Sinh viên thực hiện:

Hoàng Tiến Dũng - 19521388

Nguyễn Thành Trọng - 19522410

Giảng viên: TS. Nguyễn Vinh Tiệp

Lớp : CS331.M21

NĂM 2022

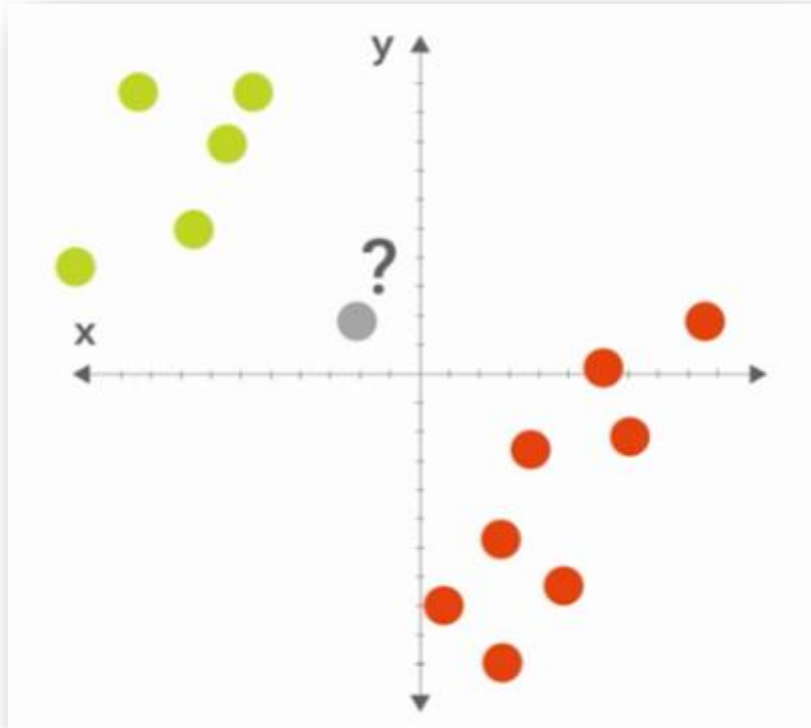
Mục lục

I. Giới thiệu	3
1. Bài toán	3
2. Ngữ cảnh	4
3. So sánh với các thuật toán khác	4
II. Phương pháp	6
1. Ý tưởng	6
2. Softmax Function	6
3. Hàm mất mát và phương pháp tối ưu	7
4. Kiểu dữ liệu	8
5. Ưu điểm và nhược điểm	9
III. Cài đặt chương trình thuật toán Softmax Regression	9
1. Tạo dữ liệu	9
2. Phân chia tập dữ liệu	11
3. Cài đặt thuật toán	11
4. Huấn luyện mô hình	13
5. Dự đoán và độ chính xác	15
IV. Thực hiện trên bộ dữ liệu thực	15
1. Dữ liệu	15
2. Phân chia bộ dữ liệu	16
3. Đánh giá	16
V. Kết luận	17
1. Softmax Regression	17
2. Source code	17
VI. Nguồn tham khảo	17

I. Giới thiệu

1. Bài toán

Bài toán phân lớp là quá trình phân lớp một đối tượng dữ liệu vào một hay nhiều lớp đã cho trước nhờ một mô hình phân lớp (model). Mô hình này được xây dựng dựa trên một tập dữ liệu được xây dựng trước đó có gán nhãn (hay còn gọi là tập huấn luyện). Quá trình phân lớp là quá trình gán nhãn cho đối tượng dữ liệu.



Hình 1: Liệu bi mới được thêm vào sẽ thuộc vào lớp bi xanh hay bi đỏ?

Vì vậy nhiệm vụ của bài toán phân lớp là cần tìm một mô hình phân lớp để khi có dữ liệu mới thì ta có thể xác định được dữ liệu đó thuộc phân lớp nào.

Có nhiều bài toán phân lớp như phân lớp nhị phân (binary), phân lớp đa lớp (multiclass), phân lớp đa trị.

Phân lớp nhị phân là bài toán gán nhãn dữ liệu cho đối tượng vào một trong hai lớp khác nhau dựa vào việc dữ liệu đó có hay không có các đặc trưng (feature) của từng lớp.

Bài toán phân lớp đa lớp là quá trình phân lớp dữ liệu với số lớp lớn hơn hai. Như vậy với từng dữ liệu thì ta phải xem xét và phân loại chúng với từng lớp khác nhau chứ không phải là hai lớp như phân lớp nhị phân.

Thực chất bài toán phân lớp nhị phân là một bài toán đặc biệt của phân lớp đa lớp.

Ứng dụng của bài toán này được sử dụng rất rộng rãi trong thực tế ví dụ như bài toán phân loại khuôn mặt, nhận dạng giọng nói, phát hiện mail spam,...

2. Ngữ cảnh

Không giống như phân lớp nhị phân, phân lớp nhiều lớp không có khái niệm về kết quả bình thường và bất thường. Thay vào đó, các mẫu được phân loại là thuộc về một trong một loạt các lớp đã biết.

Số lượng nhãn có thể rất lớn đối với một số bài toán. Ví dụ: một mô hình có thể dự đoán một bức ảnh thuộc về một trong số hàng nghìn hoặc hàng chục nghìn khuôn mặt trong hệ thống nhận dạng khuôn mặt.

Các vấn đề liên quan đến dự đoán một chuỗi từ, chẳng hạn như mô hình dịch văn bản, cũng có thể được coi là một kiểu phân loại nhiều lớp đặc biệt. Mỗi từ trong chuỗi các từ được dự đoán liên quan đến một phân loại nhiều lớp trong đó kích thước của từ vựng xác định số lượng các lớp có thể được dự đoán và có thể có kích thước hàng chục hoặc hàng trăm nghìn từ.

Người ta thường lập mô hình nhiệm vụ phân lớp nhiều lớp với một mô hình dự đoán phân phối xác suất Multinoulli cho mỗi mẫu.

Phân phối Multinoulli là một phân phối xác suất rời rạc bao gồm trường hợp một sự kiện sẽ có kết quả phân loại, ví dụ: K trong $\{1, 2, 3, \dots, K\}$. Đối với phân loại, điều này có nghĩa là mô hình dự đoán xác suất của một ví dụ thuộc về mỗi nhãn lớp.

Nhiều thuật toán được sử dụng để phân loại nhị phân có thể được sử dụng để phân loại nhiều lớp.

Các thuật toán được thiết kế để phân lớp nhị phân có thể được điều chỉnh để sử dụng cho các bài toán phân lớp nhiều lớp.

3. So sánh với các thuật toán khác

Hiện nay, khi nói về bài toán phân lớp, chúng ta sẽ tìm được rất nhiều thuật toán phục vụ cho bài toán này, trong đó mỗi thuật toán có các hướng tiếp cận, hướng xử lý khác nhau. Có thể kể tới như Logistic regression, Softmax regression, Naïve bayes, K-Nearest Neighbors, Support vector machine, Decision tree, Random forest,... Trong phần này, nhóm chúng tôi thực hiện so sánh Softmax regression với một số phương pháp phổ biến.

	Softmax Regression	Logistic Regression	Neural Network	K-Nearest Neighbors (KNN)	Support Vector Machine
--	-------------------------------	--------------------------------	---------------------------	--	---------------------------------------

Định nghĩa	Là tổng quát hóa của logistic regression trong trường hợp nhiều lớp.	Là thuật toán phân loại được dùng để gán các đối tượng cho một tập hợp giá trị rời rạc.	Là một chuỗi những thuật toán được đưa ra để tìm kiếm các mối quan hệ cơ bản trong tập hợp các dữ liệu.	Là thuật toán đi tìm đầu ra của một điểm dữ liệu mới bằng cách chỉ dựa trên thông tin của K điểm dữ liệu trong training set gần nó nhất.	Là thuật toán phân loại hoạt động bằng việc xây dựng một siêu phẳng có $(n-1)$ chiều trong không gian n chiều của dữ liệu sao cho siêu phẳng này phân loại các lớp một cách tối ưu nhất.
Đường biên	Dạng tuyến tính.	Dạng tuyến tính.	Đường biên phức tạp, là sự kết hợp phi tuyến tính thông qua các hàm kích hoạt.	Dạng phi tuyến và không smooth (trơn).	Với linear kernel đường biên có dạng tuyến tính, với các kernel khác đường biên có dạng phi tuyến.
Ưu điểm	Dễ thực hiện nhất so với các phương pháp còn lại, hoạt động tốt với dữ liệu phân biệt tuyến tính. Cho biết mức độ mà điểm dữ liệu đó thuộc về mỗi lớp.	Dễ thực hiện, dễ hiểu và hiệu quả. Độ chính xác tốt cho nhiều bộ dữ liệu đơn giản và phân biệt tuyến tính. Cho biết mức độ mà điểm dữ liệu đó thuộc về mỗi lớp.	Hiệu quả cao, khả năng tự học và điều chỉnh các trọng số để kết quả tính toán phù hợp với thực tế.	Đơn giản, dễ hiểu, nhanh và hiệu quả.	Hiệu quả trong bài toán phi tuyến, ít bị ảnh hưởng (nhạy cảm) bởi nhiễu.
Nhược điểm	Nếu kích thước bộ dữ liệu nhỏ hơn	Độ chính xác thấp nếu bộ dữ liệu quá	Cần có lượng dữ liệu đủ lớn	Cần phải chọn K bằng cách	Không phải là sự lựa chọn tốt cho

	số đặc trưng có thể dẫn tới hiện tượng overfitting. Không thực hiện tốt trên bộ dữ liệu không phân biệt tuyến tính. Thuật toán nhạy cảm với nhiễu.	nhỏ. Không thực hiện tốt trên bộ dữ liệu không phân biệt tuyến tính.	và tài nguyên tính toán để có thể đạt được kết quả tốt. Có thể xảy ra hiện tượng “overfitting” khi lượng dữ liệu quá nhỏ trong khi độ phức tạp của mô hình cao.	thủ công. Nhạy cảm với nhiễu khi K nhỏ. Việc lưu toàn bộ dữ liệu trong bộ nhớ ảnh hưởng tới hiệu năng của KNN.	bộ dữ liệu có số lượng feature lớn.
--	--	--	---	--	-------------------------------------

Bảng 1: So sánh một số thuật toán phân loại.

II. Phương pháp

1. Ý tưởng

Nhớ lại, với thuật toán Logistic Regression, output của nó là một số thập phân từ 0 đến 1.0. Ví dụ, với bài toán phân loại email rác, output của Logistic Regression bằng 0.8 cho biết 80% email đó là email rác, 20% không phải là thư rác.

Softmax Regression đã mở rộng ý tưởng của Logistic Regression cho nhiều lớp. Kết quả đầu ra cần dự đoán được dữ liệu đầu vào thuộc lớp nào, kết quả gồm 100% được chia đều cho tất cả các lớp, lớp nào có xác suất lớn nhất chính là kết quả đầu ra. Ví dụ bài toán phân loại động vật:

Chó - 0.5 → 50%

Mèo - 0.2 → 20%

Chuột - 0.3 → 30%

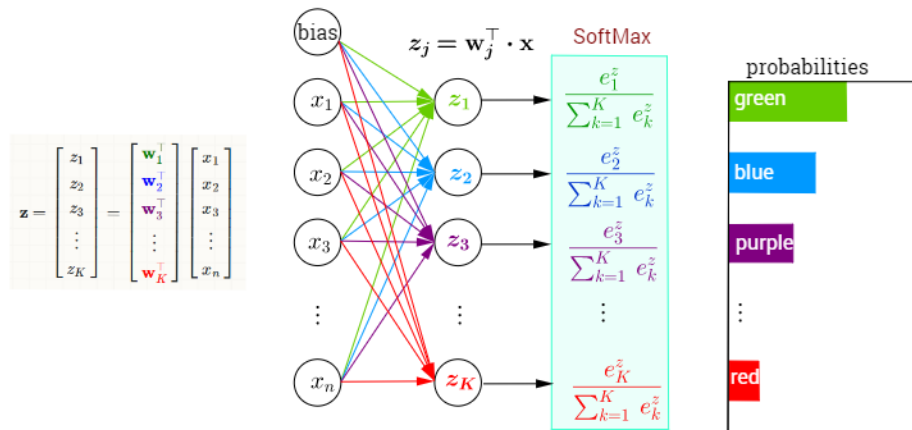
2. Softmax Function

Với output như trên yêu cầu tìm ra một hàm nào đó trả về xác suất thuộc lớp nào của input. Do đó, các giá trị cần phải dương và có tổng bằng 1.

Để có thể thỏa mãn điều kiện này, ta cần nhìn vào mọi giá trị của đầu vào, tức là đầu vào có thể mang giá trị âm. Thêm vào đó, tín hiệu của input càng mạnh thì xác suất thuộc lớp đó càng lớn tức là giá trị $z_i = w_i^T x$ càng lớn thì xác suất rơi vào lớp i càng cao. Ta có hàm softmax thỏa mãn các điều kiện trên:

$$a_i = \frac{\exp(z_i)}{\sum_{j=1}^C \exp(z_j)}, \forall i = 1, 2, 3, \dots, C$$

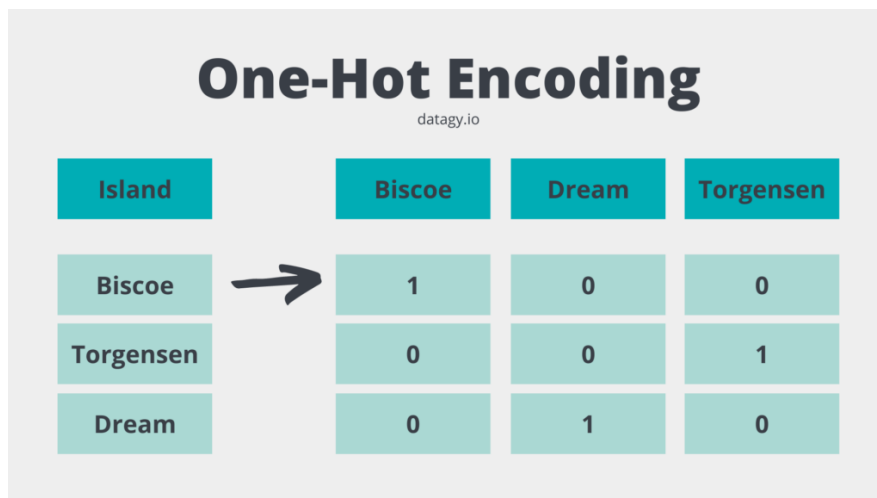
Multi-Class Classification with NN and SoftMax Function



Hình 2: Mô hình Softmax dưới dạng Neural Network.

3. Hàm mất mát và phương pháp tối ưu

Đối với cách biểu diễn như Hình 2 trên, mỗi output không còn là một giá trị tương ứng với mỗi class mà sẽ là một vector có đúng một phần tử bằng 1 và các phần tử còn lại bằng 0 trong đó phần tử bằng 1 nằm ở vị trí tương ứng với class đó. Cách mã hóa output này chính là one-hot encoding.



Hình 3: Ví dụ về One-hot encoding.

Khi đánh giá sự khác nhau giữa hai phân bố xác suất, chúng ta có một đại lượng tính hiệu quả, đó là Cross Entropy. Với Softmax Regression, ta có hàm loss giữa output dự đoán và output thực sự của một điểm dữ liệu x_i sử dụng công thức cross entropy là:

$$J(W; x_i, y_i) = - \sum_{j=1}^C y_{ji} \log(a_{ji})$$

Trong đó, a_{ji}, y_{ji} lần lượt là phần tử thứ j của vector xác suất a_i, y_i .

Kết hợp tất cả các cặp dữ liệu $x_i, y_i, i = 1, 2, \dots, N$, ta được hàm mất mát của Softmax Regression như sau:

$$J(W; X, Y) = - \sum_{i=1}^N \sum_{j=1}^C y_{ji} \log(a_{ji})$$

Đến đây, ta có thể dùng các phương pháp tối ưu hàm mất mát phổ biến như Batch Gradient Descent, Stochastic Gradient Descent (SGD), Mini-Batch Gradient Descent với ma trận trọng số W là biến cần tối ưu.

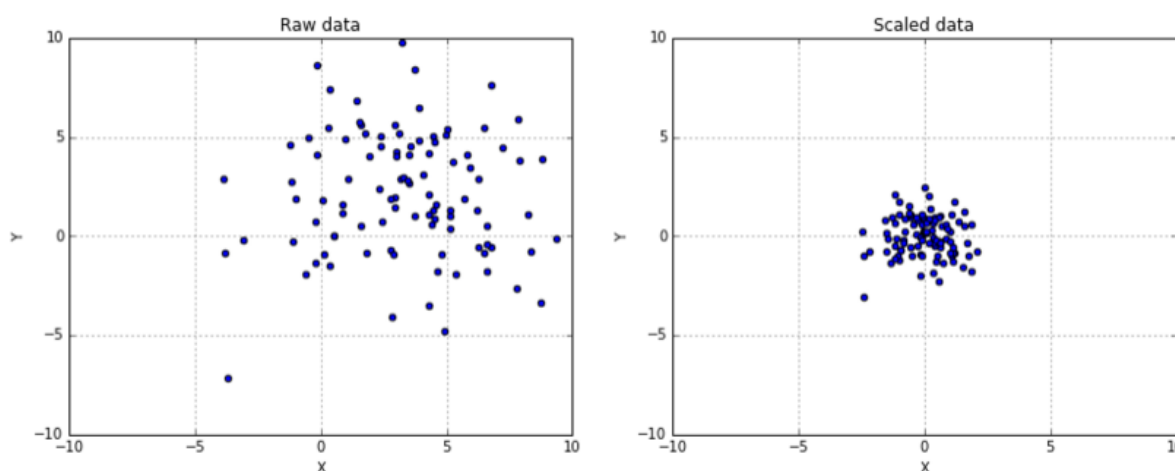
4. Kiểu dữ liệu

- Input là một vector đặc trưng đã được chuẩn hóa.
- Output là một vector trong đó mỗi phần tử thể hiện xác suất mà input đó thuộc về class nào.

Tại sao cần chuẩn hóa dữ liệu trước khi đưa vào huấn luyện?

Trong các thuật toán machine learning và deep learning, các điểm dữ liệu đôi khi được đo đạc với những đơn vị khác nhau. Hoặc có hai thành phần của vector dữ liệu chênh lệch nhau quá lớn, ví dụ một thành phần có giá trị trong khoảng 0 đến 1000, thành phần kia nằm trong khoảng từ 0 đến 1. Lúc này, chúng ta cần chuẩn hóa dữ liệu trước khi thực hiện các bước tiếp theo.

Vì các trọng số của mô hình nhỏ và được cập nhật dựa vào lỗi dự đoán nên việc chuẩn hóa dữ liệu đầu vào X là một yếu tố quan trọng. Nếu không chuẩn hóa có thể dẫn tới quá trình huấn luyện không ổn định hoặc có thể dẫn đến Exploding Gradient khiến thuật toán không chạy được.



Hình 4: Ví dụ minh họa về chuẩn hóa dữ liệu.

5. Ưu điểm và nhược điểm

- Ưu điểm:

- Softmax regression dễ thực hiện, dễ giải thích và hiệu quả.
- Độ chính xác tốt cho nhiều bộ dữ liệu đơn giản và thuật toán hoạt động tốt khi bộ dữ liệu phân biệt tuyến tính.
- Với bộ dữ liệu phân biệt tuyến tính, thuật toán thực hiện rất hiệu quả.
- Softmax regression thường được sử dụng như là layer cuối của mạng neural network trong bài toán phân loại.

- Nhược điểm:

- Nếu kích thước bộ dữ liệu nhỏ hơn số đặc trưng thì có thể dẫn tới overfitting.
- Ranh giới của Softmax regression là tuyến tính do đó với dữ liệu không phân biệt tuyến tính thuật toán thực hiện không hiệu quả.
- Sự xuất hiện của các giá trị lệch khỏi phạm vi chung của bộ dữ liệu có thể dẫn tới mô hình không chính xác vì thuật toán nhạy cảm với nhiễu.

III. Cài đặt chương trình thuật toán Softmax Regression

1. Tạo dữ liệu

Ở phần này, chúng tôi sử dụng bộ dữ liệu tự tạo. Bộ dữ liệu này xem như đã được qua xử lý và đã sẵn sàng cho việc huấn luyện. Việc sử dụng tập dữ liệu này nhằm thử nghiệm mô hình Softmax Regression mà nhóm đã cài đặt.

Trước hết, chúng tôi cài đặt và import các thư viện cần thiết cho việc cài đặt thuật toán.

```
# Import các thư viện, hàm cần thiết
from sklearn.datasets import make_blobs, make_circles
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

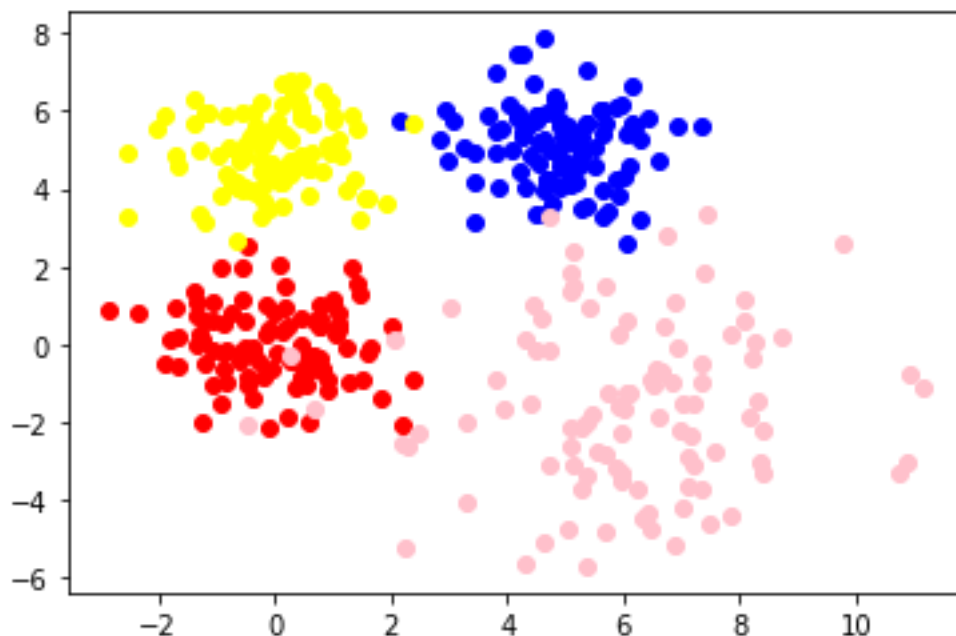
Chúng tôi đã sử dụng hàm `np.random.normal()` để tạo 4 cụm dữ liệu như hình dưới, đồng thời tạo các label tương ứng với các cụm dữ liệu đã tạo.

```
# Tạo dữ liệu
## Tạo 4 cụm dữ liệu đầu vào
xlabel0 = np.random.normal(0, 1, (2,100))
xlabel1 = np.random.normal(5, 1, (2, 100))
xlabel2 = np.random.normal(0, 1, (2, 100)) + np.array([[0], [5]])
xlabel3 = np.random.normal(6, 2, (2, 100)) + np.array([[0], [-8]])

## Tạo label tương ứng cho 4 cụm dữ liệu trên
ylabel0 = np.zeros((1, 100), np.int8)
ylabel1 = np.ones((1, 100), np.int8)
ylabel2 = np.ones((1, 100), np.int8) * 2
ylabel3 = np.ones((1,100), np.int8) * 3
```

Dữ liệu sau khi được tạo được trực quan hóa bằng cách sử dụng thư viện Matplotlib như sau.

```
# Visualize dữ liệu sau khi tạo
fig, ax = plt.subplots()
ax.scatter(xlabel0[0], xlabel0[1], color='red')
ax.scatter(xlabel1[0], xlabel1[1], color='blue')
ax.scatter(xlabel2[0], xlabel2[1], color='yellow')
ax.scatter(xlabel3[0], xlabel3[1], color='pink')
plt.show()
```



Hình 5: Trực quan hóa dữ liệu đã tạo.

Ghép các cụm dữ liệu đã tạo ở trên vào một nhóm, ta được X là ma trận các đặc trưng dữ liệu có kích thước $(2, 400)$ và y là vector label có kích thước $(400,)$.

```
# Kết hợp các dữ liệu đã tạo
X = np.concatenate((xlabel0, xlabel1, xlabel2, xlabel3), axis=1)
y = np.concatenate((ylabel0, ylabel1, ylabel2, ylabel3), axis=1)[0]
```

```
print(f"Shape of X: {X.shape}\nShape of y: {y.shape}")
```

```
Shape of X: (2, 400)
Shape of y: (400,)
```

2. Phân chia tập dữ liệu

Tiếp theo, chúng tôi sẽ phân chia tập dữ liệu thành hai tập nhỏ hơn: tập dữ liệu huấn luyện (training set) và tập dữ liệu kiểm tra (test set) với tỷ lệ 75% và 25%, `random_state = 42` nhằm đảm bảo việc phân chia dữ liệu giống nhau trong mỗi lần chạy.

```
# Chia bộ dữ liệu thành 2 bộ train và test với tỉ lệ 8:2
X_train, X_test, y_train, y_test = train_test_split(X.T, y, test_size=0.25, random_state=42)
X_train = X_train.T
X_test = X_test.T
```

Như đã nói ở trên, output của Softmax Regression và một vector one-hot. Vì thế, chúng tôi thực hiện chuyển vector `y_train` thành vector one-hot `y_train_hot`.

```
# One-hot encode cho y_train
y_train_hot = np.eye(4)[y_train]
y_train_hot = y_train_hot.T
print("Shape of y_train_hot:", y_train_hot.shape)
```

3. Cài đặt thuật toán

Để thuận tiện trong việc sử dụng và dễ dàng phát triển hơn sau này, nhóm đã quyết định cài đặt thuật toán theo hướng đối tượng.

```
class Softmax():
    """Implement Softmax Regression."""
    def __init__(self):
        pass
```

Tiếp theo đó là cài đặt hàm softmax, output của hàm softmax là một vector mà các phần tử nằm trong khoảng từ 0 đến 1 như đã nêu trong phần lý thuyết trên.

```
def softmax(self, z):
    """Cài đặt hàm softmax."""
    s = np.sum(np.exp(z), axis=0, keepdims=True) # Tính tổng của các lũy thừa cơ số e
    return np.exp(z)/s # Lấy lũy thừa cơ số e của mỗi phần tử chia cho tổng trên
```

Hàm mất mát được cài đặt theo cross-entropy. Ý tưởng của cross-entropy là nó chỉ quan tâm tới mức độ dự đoán được gán cho nhãn là chính xác hay không. Tức là, đối với nhãn thực sự là 3 thì nó chỉ quan tâm tới thành phần của y dự đoán tương ứng có bằng 3 hay không.

```
def cost_function(self, X, y, theta):
    """Cài đặt hàm mất mát."""
    m = len(X[0])
    y_pred = self.softmax(np.dot(theta.T, X)) # Tính y_pred
    cost = -1.0 * np.mean(np.sum(y * np.log(y_pred), axis = 0)) # Tính trung bình của cross entropy mỗi record
    return cost
```

Và tất nhiên không thể thiếu được hàm cập nhật theta, ở đây chúng tôi sử dụng Batch Gradient Descent để làm thuật toán cập nhật với đạo hàm của theta theo hàm loss là:

$$\frac{\partial J(W; X, Y)}{\partial W} = X(W^T X - Y)^T$$

```
def batch_gradient_descent(self, X, y, theta, iters, alpha):
    """Cập nhật theta bằng batch gradient descent."""
    m = len(X[0])
    costs = [] # Mảng costs dùng để lưu cost sau mỗi iter
    for i in range(iters):
        y_pred = self.softmax(np.dot(theta.T, X)) # Tính y_pred
        dtheta = np.dot(X, (y_pred - y).T) # Tính đạo hàm của theta theo hàm loss
        theta -= alpha * 1.0 / m * dtheta # Cập nhật theta
        cost = self.cost_function(X=X, y=y, theta=theta) # Tính và lưu cost sau mỗi iter
        costs.append(cost)
        if (i+1) % 10 == 0 or (i+1) == iters: # Cứ sau 10 iter hoặc iter cuối thì in cost ra
            print(f"Iteration: {i+1} --- cost: {cost}")
    return theta, costs
```

Hàm fit() trong class Softmax là hàm nhận vào dữ liệu X, y, số iters huấn luyện và tốc độ học learning_rate. Ở hàm này, dữ liệu đầu vào X sẽ được thêm hệ số bias, theta được khởi tạo là một vector 0 có kích thước là $(n + 1) \times c$ với n là số feature, c là số phân lớp. Hàm fit() trả về theta cuối cùng và các giá trị cost qua từng iter.

```
def fit(self, X, y, iters = 15, learning_rate=0.01):
    """Thực hiện huấn luyện mô hình."""
    m = len(X[0])
    n = len(X) # shape X = n x m
    c = len(y) # shape y = c x m
    self.X = X
    self.y = y
    self.iters = iters
    # Thêm hệ số bias cho X_train, shape X_train = (n+1) x m
    self.X_train = np.append(np.ones((1, m)), X, axis=0)
    self.y_train = self.y # shape y_train = c x m
    self.theta = np.zeros((n + 1, c)) # Tạo theta gồm các phần tử bằng 0 có shape = (n+1) x c
    # Cập nhật và trả về theta, costs
    self.theta, self.costs = self.batch_gradient_descent(self.X_train, self.y_train, self.theta, \
                                                         iters=iters, alpha=learning_rate)
    return self.theta, self.costs
```

Sử dụng thư viện Matplotlib để trực quan cost qua từng iters sau khi mô hình được huấn luyện.

```
def visual_cost(self):
    """Visual loss qua từng iter với matplotlib."""
    fig, ax = plt.subplots()
    ax.plot(range(self.iters), self.costs)
    ax.set(title='Costs', xlabel= 'Iteration', ylabel='Cost')
    plt.show()
```

Và cuối cùng là hàm predict() dùng để dự đoán, đầu vào của hàm là X_pred phải có số feature bằng với số feature của dữ liệu huấn luyện. Tham số one_hot_decode=True sẽ trả về vector phân lớp mà bản ghi tại vị trí tương ứng đạt xác suất lớn nhất, ngược lại, hàm sẽ trả ra ma trận mà mỗi phần tử là một mảng chứa các xác suất thuộc về các phân lớp mà bản ghi tại vị trí tương ứng thể hiện.

```
def predict(self, X_pred, one_hot_decode=False):
    """Dự đoán với mô hình đã được huấn luyện."""
    m = len(X_pred[0])
    n = len(self.X)
    if n != len(X_pred): # Kiểm tra chiều của X_pred có cùng chiều với X không
        print("Check your test data!")
        return
    X_pred_add_one = np.append(np.ones((1, m)), X_pred, axis=0) # Thêm hệ số bias của X_pred
    y_pred_hot = np.dot(self.theta.T, X_pred_add_one) # Dự đoán
    if one_hot_decode==True: # One-hot decode nếu muốn
        y_pred = np.argmax(y_pred_hot, axis=0)
        return y_pred
    return y_pred_hot
```

4. Huấn luyện mô hình

Khởi tạo mô hình Softmax Regression và huấn luyện trên training set với iters=1000, learning_rate=0.01. Trong khi huấn luyện, chương trình

sẽ in ra cost cứ 10 iteration một lần để người dùng có thể kiểm soát được tốc độ học của mô hình.

```
# Khởi tạo mô hình Softmax  
sm = Softmax()
```

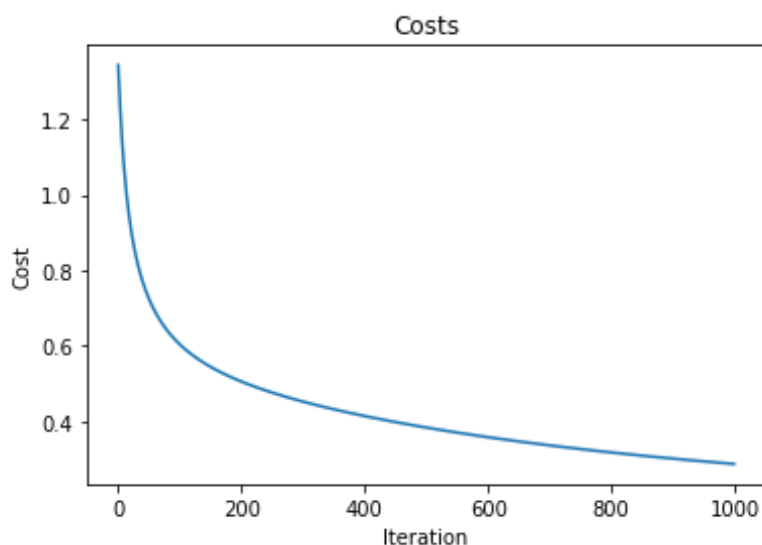
```
# Huấn luyện mô hình với iters=1000, learning_rate=0.01  
theta, costs = sm.fit(X_train, y_train_hot, 1000, 0.01)
```

```
Iteration: 10 --- cost: 1.0816454518996925  
Iteration: 20 --- cost: 0.9274602143932619  
Iteration: 30 --- cost: 0.835441521210514  
Iteration: 40 --- cost: 0.7733491524196251  
Iteration: 50 --- cost: 0.7279700935300526  
Iteration: 60 --- cost: 0.6929691267251025  
Iteration: 70 --- cost: 0.6649020796897643  
Iteration: 80 --- cost: 0.6417189892658568  
Iteration: 90 --- cost: 0.6221164122398873  
Iteration: 100 --- cost: 0.6052228972686333  
Iteration: 110 --- cost: 0.5904322522791349  
Iteration: 120 --- cost: 0.5773090054136081  
Iteration: 130 --- cost: 0.5655318588202197  
Iteration: 140 --- cost: 0.554858355172993  
Iteration: 150 --- cost: 0.5451019757238574  
Iteration: 160 --- cost: 0.5361168259177002  
Iteration: 170 --- cost: 0.5277871156576462
```

Hình 2: Cost của mô hình tại iteration tương ứng.

Trực quan hóa sự thay đổi của cost trong quá trình huấn luyện mô hình.

```
# Visual cost của mô hình sau mỗi iter  
sm.visual_cost()
```



Hình 6: Sự biến đổi của cost trong quá trình huấn luyện.

5. Dự đoán và độ chính xác

Thực hiện dự đoán trên bộ dữ liệu test set đã chia ở mục 2 trên cùng với tham số `one_hot_decode=True`, ta sẽ nhận được một vector các class mà các bản ghi ở vị trí tương ứng thuộc về.

```
# Predict X_test
y_pred = sm.predict(X_test, one_hot_decode=True)
```

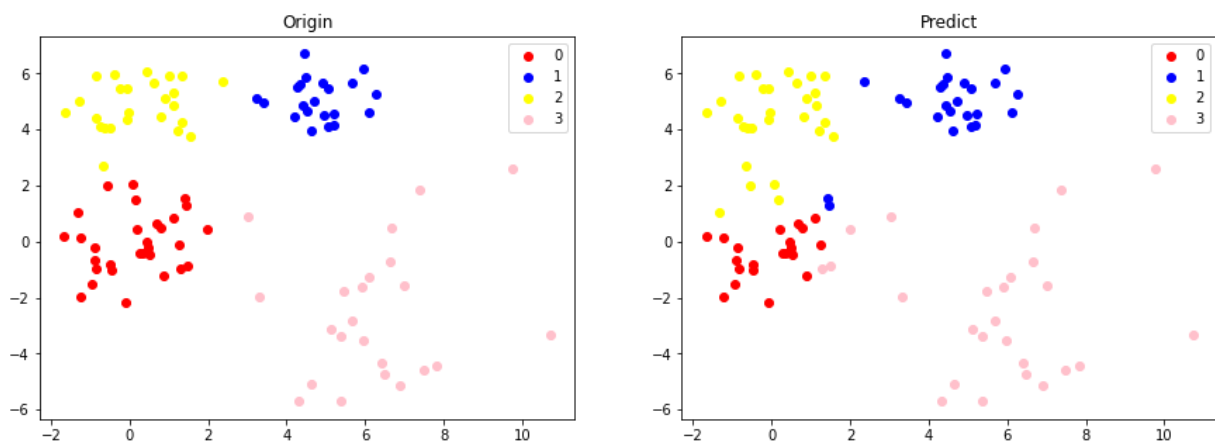
y_pred

```
array([2, 2, 0, 2, 3, 2, 3, 0, 2, 1, 0, 3, 0, 1, 1, 0, 2, 3, 2, 3, 3, 3,
       3, 2, 0, 3, 1, 2, 2, 1, 3, 1, 1, 0, 2, 1, 1, 3, 1, 1, 3, 0, 0, 0,
       1, 3, 1, 1, 2, 1, 0, 3, 3, 0, 3, 0, 2, 1, 0, 1, 1, 2, 2, 0, 1, 0,
       3, 2, 3, 0, 3, 1, 0, 0, 3, 2, 2, 3, 1, 2, 0, 0, 2, 2, 2, 2, 2, 3,
       1, 2, 2, 3, 1, 2, 1, 3, 3, 3, 2, 0])
```

Sử dụng hàm `accuracy_score` trong `sklearn.metrics` để tính độ chính xác của mô hình. Mô hình sau khi huấn luyện đạt độ chính xác 0.9 trên bộ dữ liệu test set.

```
print("Độ chính xác mô hình qua X_test là:", accuracy_score(y_test, y_pred))
```

```
Độ chính xác mô hình qua X_test là: 0.9
```



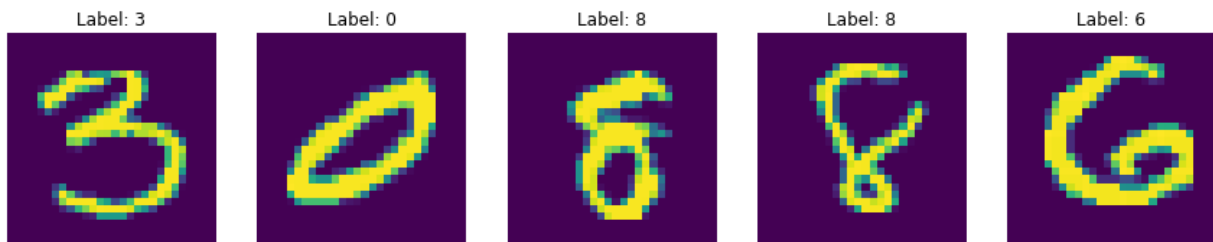
Hình 7: Bộ dữ liệu test với label thực sự và với label dự đoán từ mô hình.

IV. Thực hiện trên bộ dữ liệu thực

1. Dữ liệu

Bộ dữ liệu được sử dụng là bộ chữ số viết tay MNIST với 784 tính năng (28x28).

Bộ dữ liệu được tải về tại đây [MNIST](#), nó được chia 60.000 mẫu training và 10.000 mẫu testing. Gồm 10 lớp với 785 ($784 + 1$) tính năng (feature) cho mỗi mẫu.



Hình 8: Bộ dữ liệu MNIST.

2. Phân chia bộ dữ liệu

Như đã nói bộ dữ liệu được chia với 60000 mẫu training và 10000 mẫu testing.

Ở đây nhóm sử dụng 2 trường hợp chuẩn hóa dữ liệu khi dùng Data Standardization và dùng Data Normalization.

- Data Standardization:

```
# Chia bộ dữ liệu với 60,000 mẫu training và 10,000 mẫu testing với trường hợp Data Standardization
shuffle_index = np.random.permutation(70000)
X = mnist["data"].iloc[shuffle_index]
X_T = X.values.T
print(X.shape)
X_scaled = StandardScaler().fit_transform(X_T)
Y = mnist["target"].iloc[shuffle_index]
x_train_sc, x_test_sc, y_train_sc, y_test_sc = X_scaled[:, :60000], X_scaled[:, 60000:], Y[:60000].astype(int), Y[60000:].astype(int)
```

- Data Normalization:

```
# Chia bộ dữ liệu với 60,000 mẫu training và 10,000 mẫu testing với trường hợp Data Normalization
shuffle_index = np.random.permutation(70000)
X = mnist["data"].iloc[shuffle_index]/255
X_T = X.values.T
print(X.shape)
Y = mnist["target"].iloc[shuffle_index]
x_train, x_test, y_train, y_test = X_T[:, :60000], X_T[:, 60000:], Y[:60000].astype(int), Y[60000:].astype(int)
```

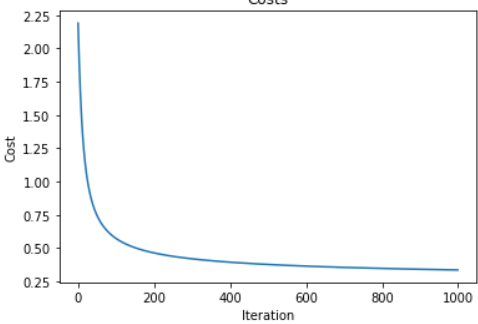
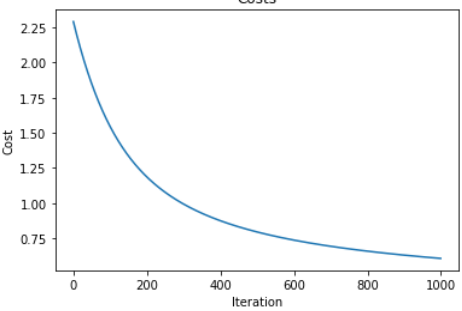
Sau đó mã hóa one-hot label:

```
# One-hot encode cho y_train
y_train_hot = np.eye(10)[y_train]
y_train_hot = y_train_hot.T
```

```
# One-hot encode cho y_train_sc
y_train_sc_hot = np.eye(10)[y_train_sc]
y_train_sc_hot = y_train_sc_hot.T
```

3. Đánh giá

Huấn luyện mô hình với $\text{iters}=1000$, $\text{learning_rate}=0.01$ với các cách chuẩn hóa dữ liệu khác nhau ta có kết quả sau:

	Data Standardization	Data Normalization
Training cost	0.336	0.605
Training process		
Training time	570.5s	586.1s
Accuracy	0.9052	0.8605

Bảng 2: Đánh giá thuật toán trên bộ dữ liệu thực tế với các cách chuẩn hóa.

V. Kết luận

1. Softmax Regression

- Softmax Regression là thuật toán tổng quát hóa của Logistic Regression trong trường hợp nhiều lớp.
- Là một trong những thuật toán phổ biến nhất. Đặc biệt, Softmax Regression được sử dụng nhiều trong các mạng Neural nhiều lớp, trong đó những lớp phía trước được coi như một bộ Feature Extractor, lớp cuối cùng là Softmax Regression.

2. Source code

- Cài đặt thuật toán:
https://colab.research.google.com/drive/15qGr_b4V0KsLDMXP45oLTK_ZXLyEmXt?usp=sharing
- Áp dụng trên bộ dữ liệu thực:
https://colab.research.google.com/drive/1rAp25rRBH3_bsnzXomNdzl2uQvMeoz7E?usp=sharing

VI. Nguồn tham khảo

- [1] <https://machinelearningcoban.com/2017/02/17/softmax/>
- [2] <https://datagy.io/sklearn-one-hot-encode/>
- [3] [Multiclass classification with softmax regression and gradient descent](#)
- [4] <https://www.mygreatlearning.com/blog/introduction-to-softmax-regression/>

[5] <https://www.kaggle.com/code/synnfusion/softmax-on-mnist-from-scratch/notebook>

[6] <https://towardsdatascience.com/softmax-function-simplified-714068bf8156>