ĐẠI HỌC QUỐC GIA HÀ NỘI

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

BÁO CÁO BÀI TẬP LỚN

MACHINE LEARNING MODELS: SUPERVISED (PARAMETERIC) NEURAL NETWORK NAIVE BAYES CLASSIFIER

Họ và tên sinh viên: Dương Huy Anh Vũ - 21021393

Nguyễn Thanh Tùng - 21021390

Phạm Quang Vinh – 21021391

Lớp học phần: INT3401_ 20

Giảng viên hướng dẫn: TS. Trần Hồng Việt

HÀ NỘI - 2023

Mục lục

I. Giới thiệu về Supervised Learning (Học có giám sát)

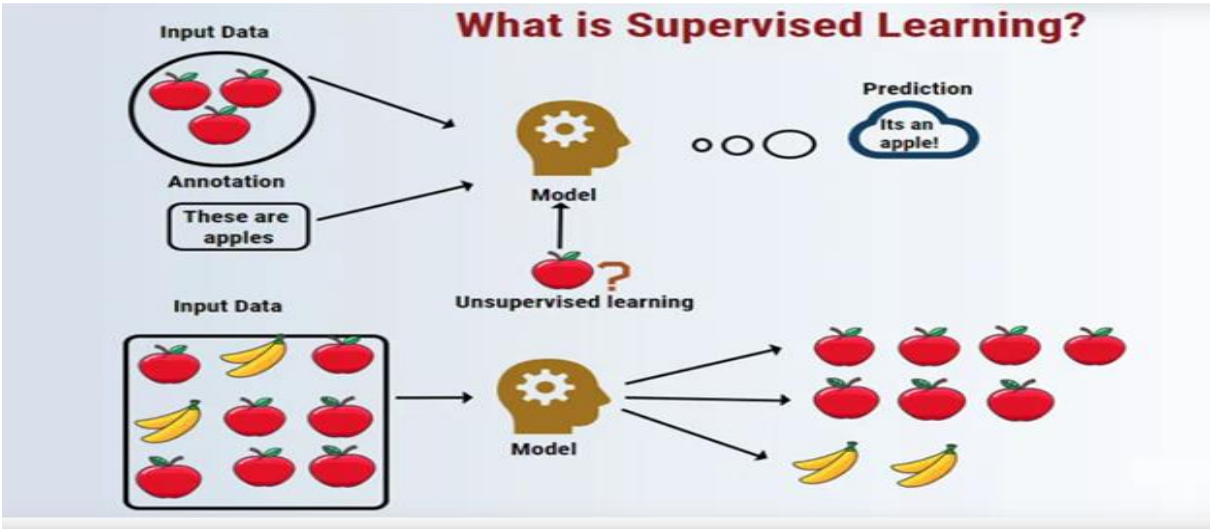
1. Khái niệm của Supervised learning

- Định nghĩa : Supervised learning là một tập hợp con cụ thể của Machine learning và Artifical intelligence . Nó dựa vào các tập dữ liệu được gắn nhãn để đào tạo các thuật toán, cho phép phân loại hoặc dự đoán kết quả chính xác.

- Quá trình : Dữ liệu đầu vào cùng với các nhãn tương ứng được đưa vào mô hình. Thông qua các lần lặp lại, mô hình sẽ điều chỉnh các tham số (trọng số) bên trong của nó để giảm thiểu lỗi, thường sử dụng các kỹ thuật như xác thực chéo để đảm bảo hiệu suất tối ưu.

- Ứng dụng: Supervised learning được sử dụng rộng rãi trong việc giải quyết các thách thức trong thế giới thực trên quy mô lớn. Một ví dụ nổi bật là tính năng lọc email, trong đó tính năng này phân loại chính xác email là thư rác hoặc không phải thư rác, tạo điều kiện cho việc tổ chức hộp thư đến tốt hơn.

- Độ chính xác và xây dựng mô hình: Bằng cách tận dụng dữ liệu được gắn nhãn trong quá trình đào tạo, Supervised learning cho phép tạo ra các mô hình học máy có độ chính xác cao. Độ chính xác này rất quan trọng đối với các ứng dụng mà việc phân loại hoặc dự đoán chính xác là điều tối quan trọng.



2. Nguyên lý hoạt động

- Supervised learning sử dụng một tập dữ liệu huấn luyện để chỉ dẫn các mô hình trong việc sản xuất đầu ra mong muốn. Trong tập dữ liệu huấn luyện này, cả đầu vào và đầu ra chính xác đều được bao gồm, cho phép mô hình dần dần học. Độ chính xác của thuật toán được đo lường thông qua hàm mất mát, được điều chỉnh cho đến khi sai số được giảm một cách phù hợp. Quá trình lặp này nâng cao khả năng của mô hình tạo ra kết quả chính xác. Cụ thể, Supervised learning thường hoạt động thông qua một loạt các bước phương pháp.

- Thu thập Dữ liệu: Bước đầu tiên là thu thập một tập dữ liệu bao gồm các cặp đầu vào-đầu ra. Tập dữ liệu này phục vụ như tập huấn luyện.

- Tiền xử lý Dữ liệu: Dữ liệu được thu thập sau đó được làm sạch và tiền xử lý. Điều này bao gồm loại bỏ nhiễu hoặc dữ liệu không liên quan, xử lý dữ liệu bị thiếu và có thể thay đổi tỷ lệ và chuẩn hóa dữ liệu.

- Lựa chọn Mô hình: Dựa trên bản chất của dữ liệu và vấn đề cụ thể, một mô hình hoặc thuật toán phù hợp được lựa chọn, như hồi quy tuyến tính, cây quyết định hoặc mạng Neural Network.

- Thiết lập Mô hình: Mô hình sau đó được thiết lập trên dữ liệu đã được tiền xử lý. Mô hình học bằng cách điều chỉnh dữ liệu đầu vào thành đầu ra tương ứng. Nó điều chỉnh các tham số nội tại của mình để giảm thiểu sự khác biệt, hoặc "sai số", giữa các dự đoán của nó và đầu ra thực tế.

- Đánh giá: Khi mô hình được huấn luyện, nó được đánh giá bằng một tập dữ liệu riêng biệt, được gọi là tập kiểm tra hoặc thử nghiệm. Dữ liệu này không được sử dụng trong giai đoạn huấn luyện và phục vụ để đánh giá mức độ mà mô hình có thể tổng quát hóa những gì nó đã học cho dữ liệu mới, không được nhìn thấy trước.

- Tối ưu hóa: Nếu hiệu suất của mô hình không đạt yêu cầu, các tham số được điều chỉnh và mô hình được huấn luyện lại. Quá trình này tiếp tục cho đến khi hiệu suất của mô hình đạt đến một mức chấp nhận được.

- Dự đoán: Cuối cùng, mô hình đã được huấn luyện được sử dụng để đưa ra dự đoán trên dữ liệu mới, không được nhìn thấy trước.

- Một số ví dụ phổ biến của thuật toán học máy có giám sát là:

  + Hồi quy tuyến tính cho các vấn đề hồi quy.

  + Nguyên lý “Khu rừng ngẫu nhiên” cho việc phân loại và hồi quy.

  + Hỗ trợ các hệ máy vector cho các vấn đề về phân loại.

3. Ứng dụng

Nhận dạng hình ảnh và đối tượng: Các thuật toán Supervised learning có thể được sử dụng để xác định vị trí, khoanh vùng và phân loại các đối tượng ra khỏi video hoặc hình ảnh, điều này rất hữu ích khi áp dụng vào các kỹ thuật thị giác máy tính và phân tích hình ảnh khác nhau.

Phân tích dự đoán: Một trường hợp sử dụng rộng rãi cho các mô hình học tập có giám sát là tạo hệ thống phân tích dự đoán để cung cấp thông tin chi tiết về các điểm dữ liệu kinh doanh khác nhau. Điều này cho phép doanh nghiệp dự đoán các kết quả dựa trên một biến đầu ra nhất định, giúp các lãnh đạo chứng minh tính chính xác của các quyết định hoặc các vấn đề xoay quanh lợi ích của doanh nghiệp.

Phân tích tâm lý khách hàng: Sử dụng các thuật toán học máy có giám sát, các tổ chức có thể trích xuất và phân loại các thông tin quan trọng từ lượng lớn dữ liệu (bao gồm ngữ cảnh, cảm xúc và ý định). Điều này vô cùng hữu ích khi doanh nghiệp có thể hiểu rõ hơn các tương tác của khách hàng.

Phát hiện thư rác: Phát hiện thư rác là một ví dụ khác về mô hình Supervised learning. Sử dụng các thuật toán phân loại có giám sát, các tổ chức có thể đào tạo cơ sở dữ liệu để nhận ra các mẫu hoặc điểm bất thường trong dữ liệu mới để tổ chức các thư từ liên quan đến thư rác và không phải thư rác một cách hiệu quả.

II. Neural Network (Mạng thần kinh nhân tạo)

1. Định nghĩa về Neural Network

Định nghĩa :

- Neural Network, hay còn gọi là mạng Neural Network nhân tạo hay mạng lưới thần kinh nhân tạo, là một mô hình toán học phức tạp được phát triển dựa theo các mạng Neural Network sinh học. Cụ thể hơn, Neural Network được xây dựng dựa theo mô hình hoạt động của các tế bào thần kinh của con người.

- Ở não người, các dây thần kinh kết nối các nút, gọi là tế bào thần kinh, lại với nhau. Còn ở Neural Network, các nút này được gọi là Neural Network nhân tạo. Việc kết nối các điểm này lại sẽ tạo ra một hệ thống dây chằng chịt, khi đó nó được gọi là mạng Neural Network nhân tạo.

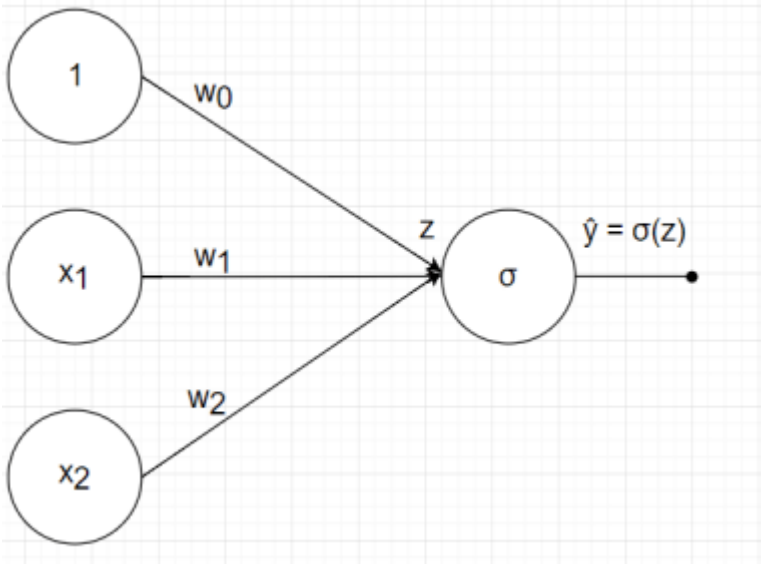
- Tương tự như hoạt động não bộ ở con người, bao gồm việc tiếp nhận thông tin, xử lý thông tin và hành động, Neural Network cũng được tạo ra để thực hiện những thao tác này. Tuy nhiên, mạng Neural Network nhân tạo sẽ sử dụng các thuật toán để xác định và phân tích mối quan hệ trong tệp dữ liệu mà chúng cần giải quyết.

2. Kiến trúc mạng của Neural Network

- Logistic regression: Logistic regression là mô hình neural network đơn giản nhất chỉ với input layer và output layer.

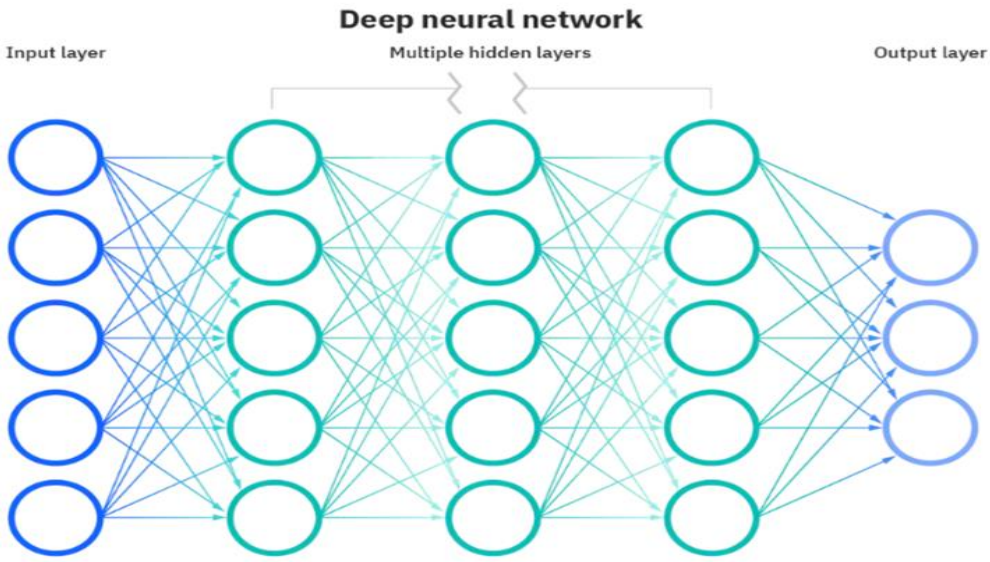
+ Mô hình của logistic regression là: $y^=\sigma(w0+w1*x1+w2*x2)$. Có 2 bước:

- Tính tổng linear: $z=1*w0+x1*w1+x2*w2z=1*w0+x1*w1+x2*w2$
- Áp dụng sigmoid function: $y^=\sigma(z)y^=\sigma(z)$



Hệ số $w0$ được gọi là bias và hàm sigmoid ở đây được gọi là **activation function**.

Mạng Neural Network được cấu thành từ những tầng perceptron, gồm 3 tầng chính:
- Tầng vào (input layer): Như trong hình ở dưới, tầng này nằm bên phía trái, thể hiện cho các dữ liệu đầu vào.
- Tầng ra (output layer): Ngược lại với tầng vào, tầng ra thể hiện cho đầu ra của mạng Neural Network và nằm bên phía bên phải của hình.
- Tầng ẩn (hidden layer): Đây là tầng nằm ở giữa, thể hiện cho quá trình xử lý thông tin và suy luận của mạng. Nó sẽ nhận các thông tin đầu vào ở đầu vào và trả kết quả ở đầu ra thông qua chức năng kích hoạt.
- Các hình tròn được gọi là node.
- Mỗi mô hình luôn có 1 input layer, 1 output layer, có thể có hoặc không các hidden layer. Tổng số layer trong mô hình được quy ước là số layer – 1 (Không tính input layer).
- Mỗi node trong hidden layer và output layer :
  - Liên kết với tất cả các node ở layer trước đó với các hệ số w riêng.
  - Mỗi node có 1 hệ số bias b riêng.
  - Diễn ra 2 bước: tính tổng linear và áp dụng activation function.



- Kí hiệu:

Số node trong hidden layer thứ i là $l(i)l(i)$.

Ma trận $W(k)$ kích thước $l(k-1)*l(k)$ là ma trận hệ số giữa layer (k-1) và layer k, trong đó $wij(k)$ là hệ số kết nối từ node thứ i của layer k-1 đến node thứ j của layer k.

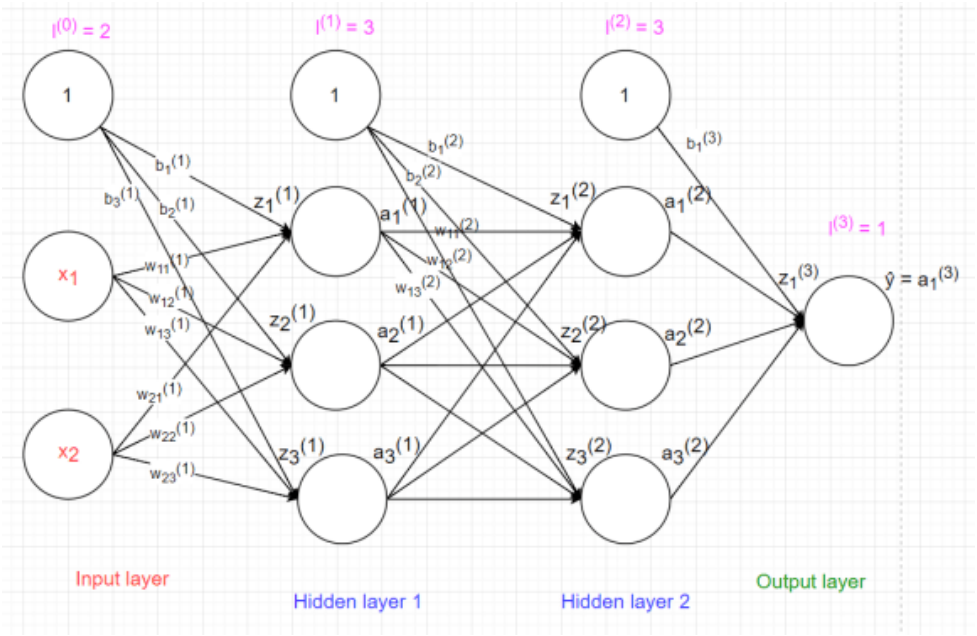
Vector $b(k)$ kích thước $lk*1$ là hệ số bias của các node trong layer k, trong đó $bi(k)$ là bias của node thứ i trong layer k.

Với node thứ i trong layer l có bias $bi(l)$ thực hiện 2 bước:

- Tính tổng linear: $zi(l)=\sum j=1l(l-1)aj(l-1)*wji(l)+bi(l)$, là tổng tất cả các node trong layer trước nhân với hệ số w tương ứng, rồi cộng với bias b.
- Áp dụng activation function: $ai(l)=\sigma(zi(l))$

Vector $z(k)$ kích thước $l(k)*1$ là giá trị các node trong layer k sau bước tính tổng linear.

Vector $a(k)$ kích thước $l(k)*1$ là giá trị của các node trong layer k sau khi áp dụng hàm activation function.



Mô hình neural network trên gồm 3 layer. Input layer có 2 node ($l(0)=2l(0)=2$), hidden layer 1 có 3 node, hidden layer 2 có 3 node và output layer có 1 node.

Do mỗi node trong hidden layer và output layer đều có bias nên trong input layer và hidden layer cần thêm node 1 để tính bias (nhưng không tính vào tổng số node layer có).

Tại node thứ 2 ở layer 1, ta có:

- $z2(1)=x1*w12(1)+x2*w22(1)+b2(1)$
- $a2(1)=\sigma(z2(1))$

Tại node thứ 2 ở layer 1, ta có:

- $z2(1)=x1*w12(1)+x2*w22(1)+b2(1)$
- $a2(1)=\sigma(z2(1))$

- Feedforward (Lan truyền tiến)

Gọi input layer là $a(0)(=x)a(0)(=x)$ kích thước 2*1.

$$z^{(1)} = \begin{bmatrix} z_1^{(1)} \\ z_2^{(1)} \\ z_3^{(1)} \end{bmatrix} = \begin{bmatrix} a_1^{(0)} * w_{11}^{(1)} + a_2^{(0)} * w_{21}^{(1)} + a_3^{(0)} * w_{31}^{(1)} + b_1^{(1)} \\ a_1^{(0)} * w_{12}^{(1)} + a_2^{(0)} * w_{22}^{(1)} + a_3^{(0)} * w_{32}^{(1)} + b_2^{(1)} \\ a_1^{(0)} * w_{13}^{(1)} + a_2^{(0)} * w_{23}^{(1)} + a_3^{(0)} * w_{33}^{(1)} + b_3^{(1)} \end{bmatrix}$$

$$= (W^{(1)})^T * a^{(0)} + b^{(1)}$$

$$a^{(1)} = \sigma(z^{(1)})$$

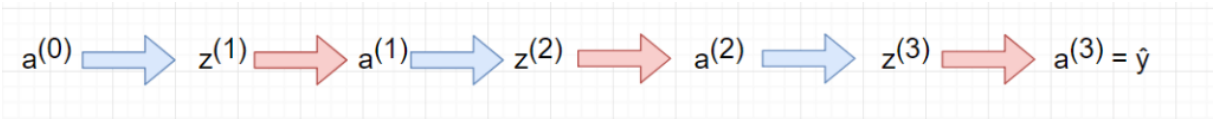
Tương tự ta có:

$z(2)=(W(2))T*a(1)+b(2)$

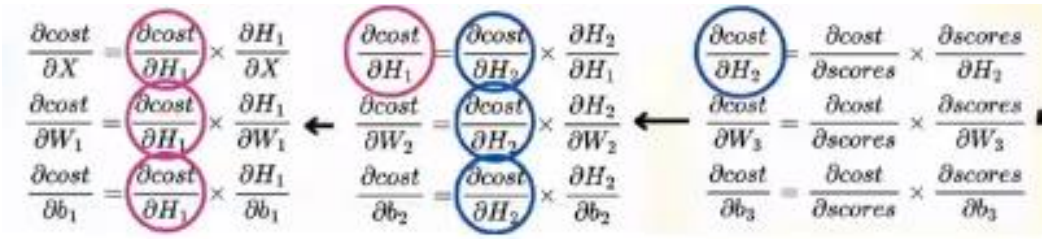
$a(2)=\sigma(z(2))$

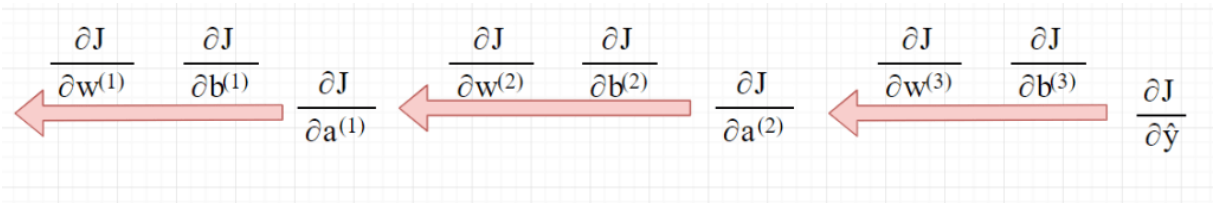
$z(3)=(W(3))T*a(2)+b(3)$

$y\hat{}=a(3)=\sigma(z(3))$



- Backward pass (Lan truyền ngược)





3. Tầm quan trọng của Neural Network.

- Mạng Neural Network có thể giúp máy tính đưa ra các quyết định thông minh chỉ với sự hỗ trợ hạn chế của con người. Lý do là vì chúng có thể học hỏi và dựng mô hình các mối quan hệ giữa dữ liệu đầu vào và đầu ra phi tuyến tính, phức tạp. Ví dụ, chúng có thể đảm nhận những nhiệm vụ sau.

- Đưa ra các khái quát hoặc suy luận:
Mạng Neural Network có thể hiểu rõ dữ liệu phi cấu trúc và đưa ra các nhận xét chung mà không cần đào tạo cụ thể. Chẳng hạn, chúng có thể nhận ra hai câu đầu vào khác nhau có ý nghĩa tương tự nhau:

· Bạn có thể chỉ cho tôi cách thanh toán không?
· Tôi có thể chuyển tiền bằng cách nào?

Mạng Neural Network sẽ biết rằng cả hai câu này đều có chung ý nghĩa. Hoặc chúng sẽ có thể phân biệt được đại khái rằng Baxter Road là một địa điểm, còn Baxter Smith là tên người.

4. Phân loại Neural Network.

Mạng Neural Network nhân tạo có thể được phân loại theo phương thức dữ liệu được truyền từ nút đầu vào đến nút đầu ra. Dưới đây là một số ví dụ:

- Mạng Neural Network truyền thẳng

Mạng Neural Network truyền thẳng xử lý dữ liệu theo một chiều, từ nút đầu vào đến nút đầu ra. Mỗi nút trong một lớp được kết nối với tất cả các nút trong lớp tiếp theo. Mạng truyền thẳng sử dụng một quy trình phản hồi để cải thiện dự đoán theo thời gian.

- Mạng Neural Network nhân tạo liên tục học hỏi bằng cách sử dụng vòng lặp phản hồi hiệu chỉnh để cải thiện phân tích dự đoán của chúng. Đơn giản mà nói, bạn có thể coi rằng dữ liệu truyền từ nút đầu vào đến nút đầu ra qua nhiều lối đi khác nhau trong mạng Neural Network. Chỉ có duy nhất một lối đi chính xác, ánh xạ nút đầu vào đến nút đầu ra thích hợp. Để tìm ra lối đi này, mạng Neural Network sử dụng một vòng lặp phản hồi với cách thức hoạt động như sau:

1. Mỗi nút đưa ra một dự đoán về nút tiếp theo trên lối đi.

2. Nút này sẽ kiểm tra tính chính xác của dự đoán. Các nút sẽ chỉ định giá trị trọng số cao hơn cho những lối đi tới nhiều dự đoán chính xác hơn và giá trị trọng số thấp hơn cho các lối đi tới dự đoán không chính xác.

3. Đối với điểm dữ liệu tiếp theo, các nút đưa ra dự đoán mới bằng cách sử dụng các lối đi có trọng số cao hơn rồi lặp lại Bước 1.

- Mạng Neural Network tích chập

Những lớp ẩn trong mạng Neural Network tích chập thực hiện các chức năng toán học cụ thể, như tóm tắt hoặc sàng lọc, được gọi là tích chập. Chúng rất hữu ích trong việc phân loại hình ảnh vì chúng có thể trích xuất các đặc điểm liên quan từ hình ảnh, điều này có lợi cho việc nhận dạng và phân loại hình ảnh. Biểu mẫu mới dễ xử lý hơn mà không làm mất đi các đặc điểm quan trọng để đưa ra dự đoán chính xác. Mỗi lớp ẩn trích xuất và xử lý các đặc điểm hình ảnh khác nhau, như các cạnh, màu sắc và độ sâu.

5. Cách để training Neural Network

Đạo tào mạng Neural Network là quy trình dạy mạng Neural Network thực hiện một nhiệm vụ. Mạng Neural Network học hỏi bằng cách xử lý ban đầu một số các tập hợp dữ liệu lớn đã được hoặc chưa được gắn nhãn. Bằng cách sử dụng những ví dụ này, chúng có thể xử lý các dữ liệu đầu vào chưa xác định một cách chính xác hơn.

Trong Supervised learning, các nhà khoa học dữ liệu đưa cho mạng Neural Network nhân tạo các tập dữ liệu đã gắn nhãn để cung cấp trước câu trả lời đúng. Ví dụ: một mạng deep learning được đào tạo về nhận diện khuôn mặt ban đầu xử lý hàng trăm nghìn hình ảnh về khuôn mặt người, với các thuật ngữ khác nhau liên quan đến sắc tộc, quốc tịch hoặc cảm xúc mô tả mỗi hình ảnh.
Mạng Neural Network dần tích lũy kiến thức từ các tập dữ liệu cung cấp trước câu trả lời đúng này. Sau khi đã được đào tạo, mạng bắt đầu đưa ra phỏng đoán về sắc tộc hoặc cảm xúc của một hình ảnh khuôn mặt người mới mà nó chưa từng xử lý trước đây.

6. Ứng dụng

Hiện nay, Neural Network được ứng dụng rộng rãi trong nhiều lĩnh vực như kinh doanh, giáo dục, y tế, công nghệ thông tin, công nghệ blockchain, … Dưới đây là một số ứng dụng quen thuộc của Neural Network:

- Nhận dạng chữ viết tay
  Hiện nay có rất nhiều website hay phần mềm giúp con người tạo ra chữ ký online. Trong quá trình này, các ký tự viết tay sẽ được chuyển đổi thành các ký tự kỹ thuật số bởi mạng nơ-ron nhân tạo.

- Nén hình ảnh
  Neural Network được sử dụng nhiều trong việc lưu trữ, mã hóa và tạo nén hình ảnh. Con người có thể tái tạo và tối ưu kích thước dữ liệu bằng cách sử dụng Neural Network. Việc này không những giúp chúng ta tiết kiệm bộ nhớ mà còn gửi thông tin nhanh hơn.

- Tối ưu quãng đường di chuyển
  Ứng dụng này nổi bật trong các phần mềm bản đồ và điển hình nhất là Google Map. Khi chúng ta tìm kiếm một địa điểm cụ thể nào đó và đường đi đến đó, Google sẽ đề xuất 2-3 con đường cho mình chọn. Trong đó, con đường ngắn nhất và tối ưu nhất sẽ được làm nổi bật.

- Dự đoán giao dịch chứng khoán
  Với sự thay đổi nhanh chóng và khó hiểu của sàn giao dịch chứng khoán, Neural Network đã được tận dụng để dự báo những xê dịch trong thị trường. Tại đây, Neural Network sẽ kiểm tra và phân tích hàng loạt các yếu tố ảnh hưởng đến thị trường để đưa ra các dự đoán hàng ngày, giúp các nhà môi giới có được những quyết định chính xác hơn và ít rủi ro hơn.

- Digital Marketing
  Trong các năm đổ lại đây, việc ứng dụng trí tuệ nhân tạo trong Digital Marketing không còn quá xa lạ đối với dân kinh doanh. Việc sử dụng Deep Learning và Neural Network trong việc tiếp thị giúp mang lại trải nghiệm tốt hơn cho khách hàng và tăng doanh thu cho doanh nghiệp. Một số ví dụ điển hình như hệ thống Chatbot, xây dựng nội dung cá nhân hóa, phân tích người dùng bằng cách nhận diện hình ảnh, …

7. Xây dựng Neural Network

- Data loader: Tập dữ liệu MNIST bao gồm 70.000 hình ảnh chữ số viết tay từ 0 đến 9; 60.000 để huấn luyện (train set) và 10.000 để kiểm tra (test set)

```
import numpy as np
import struct
from array import array
from os.path  import join
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

'''
Define a class for loading the MNIST dataset from the files
Source: https://www.kaggle.com/code/hojjatk/read-mnist-dataset/notebook
'''
class MnistDataloader(object):
    def __init__(self, training_images_filepath, training_labels_filepath,
                 test_images_filepath, test_labels_filepath):
        '''
        Set the files path
        TRAIN_IMAGES_FILE_PATH:  Path to the train images
        training_labels_filepath: Path to the train labels
        test_images_filepath: Path to the test images
        test_labels_filepath: Path to the test labels
        '''
        self.training_images_filepath = training_images_filepath
        self.training_labels_filepath = training_labels_filepath
        self.test_images_filepath = test_images_filepath
        self.test_labels_filepath = test_labels_filepath

    def read_images_labels(self, images_filepath, labels_filepath):
        '''
        Read the images and labels
        images_filepath: Path to the images
        labels_filepath: Path to the labels
        Return: The numpy array containing the images and labels
        '''
        labels = []
        with open(labels_filepath, 'rb') as file:
            magic, size = struct.unpack(">II", file.read(8))
            if magic != 2049:
```

```
            raise ValueError('Magic number mismatch, expected 2049, got
{}'.format(magic))
        labels = array("B", file.read())

    with open(images_filepath, 'rb') as file:
        magic, size, rows, cols = struct.unpack(">IIII", file.read(16))
        if magic != 2051:
            raise ValueError('Magic number mismatch, expected 2051, got
{}'.format(magic))
        image_data = array("B", file.read())

    images = []
    for i in range(size):
        images.append([0] * rows * cols)

    for i in range(size):
        img = np.array(image_data[i * rows * cols:(i + 1) * rows * cols])
        img = img.reshape(28 *28)
        images[i][:] = img

    return np.array(images), np.array(labels)

  def load_data(self):
    '''
    Load the train and test images, train and test labels from the file paths
    '''
    x_train, y_train = self.read_images_labels(self.training_images_filepath,
self.training_labels_filepath)
    x_test, y_test = self.read_images_labels(self.test_images_filepath,
self.test_labels_filepath)
    return (x_train, y_train), (x_test, y_test)


# Define the paths of the files
INPUT_PATH  = 'data'
TRAIN_IMAGES_FILE_PATH  = join(INPUT_PATH,  'train-images.idx3-ubyte')
TRAIN_LABELS_FILE_PATH  = join(INPUT_PATH,  'train-labels.idx1-ubyte')
TEST_IMAGES_FILE_PATH  = join(INPUT_PATH,  'test-images.idx3-ubyte')
TEST_LABELS_FILE_PATH  = join(INPUT_PATH,  'test-labels.idx1-ubyte')

# Create MNISTDataloader and load the images
mnist = MnistDataloader(
  training_images_filepath=TRAIN_IMAGES_FILE_PATH,
  training_labels_filepath=TRAIN_LABELS_FILE_PATH,
  test_images_filepath=TEST_IMAGES_FILE_PATH,
```

```
    test_labels_filepath=TEST_LABELS_FILE_PATH
)

# Split into train and test set
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# Split into train and validation set
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.05,
stratify=y_train)

# Standardize the train, validation and test dataset
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_val = scaler.transform(X_val)
X_test = scaler.transform(X_test)
```

- Các lớp (Layer):

```
import numpy as np


"""
Define a class for the linear layer
"""
class Linear:
    def __init__(self, in_features, out_features, regularization=0.0, learning_rate=3e-3):
        """
        Initialize the model's parameters
        """
        # Create the weight (with Xavier initialization) and bias
        self.W = np.random.randn(in_features, out_features) / 10 # * np.sqrt(2 /
in_features)
        self.b = np.zeros((out_features,))
        self.reg = regularization
        self.lr = learning_rate

    def forward(self, X):
        """
        Perform the forward pass of the model
        Cache the result for the backward pass
        """
        self.cache = (X,)
        result = X @ self.W + self.b
        return result
```

```
    def backward(self, dLoss):
        """
        Perform the backward pass of the model
        Calculate the gradients and updates the weight and bias
        """
        # Get the previous input
        X = self.cache[0]

        # Calculate the gradients
        dW = X.T @ dLoss + self.reg * self.W
        db = np.sum(dLoss, axis=0) + self.reg * self.b
        dX = dLoss @ self.W.T

        # Update the weight and bias
        self.W -= self.lr * dW
        self.b -= self.lr * db

        # Return the gradients
        return dX, dW, db


"""
Define a class for the ReLU layer
"""
class ReLU:
    def forward(self, X):
        """
        Perform the forward pass of the ReLU layer
        Cache the result for the backward pass
        """
        self.cache = (X,)
        result = np.maximum(X, 0)
        return result

    def backward(self, dLoss):
        """
        Perform the backward pass of the ReLU layer
        """
        # Get the input from cache
        X = self.cache[0]

        # Calculate the gradient
        mask = (X > 0).astype(np.float32)
        dX = dLoss * mask
```

```
        # Return the gradient
        return dX


"""
Define a layer that combines the Linear layer and the ReLU layer
"""
class LinearAndReLU:
    def __init__(self, in_features, out_features, regularization, learning_rate):
        """
        Create two layers
        """
        self.linear = Linear(in_features, out_features, regularization, learning_rate)
        self.relu = ReLU()

    def forward(self, X):
        """
        Perform the forward pass on the two layers
        """
        hidden = self.linear.forward(X)
        return self.relu.forward(hidden)

    def backward(self, dLoss):
        """
        Perform the backward pass of the two layers
        """
        dHidden = self.relu.backward(dLoss)
        dX, dW, db = self.linear.backward(dHidden)
        return dX, dW, db


"""
Define a Softmax layer
"""
class Softmax:
    def forward(self, scores, y):
        """
        Perform the forward pass of the Softmax layer
        Cache the result for the backward pass
        """
        # Get the dimension
        N, C = scores.shape

        # Exponential
```

```
        E = np.exp(scores)

        # Distribution
        D = E / np.sum(E, axis=1).reshape(-1, 1)

        # Choose the probabability
        P = D[np.arange(N), y]
        self.cache = (D,)

        # Calculate the log probability
        L = -np.log(P)

        # Calculate the cost
        cost = (1 / N) * np.sum(L)
        return cost

    def backward(self, y):
        """
        Perform the backward pass of the Softmax layer
        """
        # Get the cache
        D = self.cache[0]

        # Get the dimension
        N, C = D.shape

        # Calculate the gradient
        dScores = D
        dScores[np.arange(N), y] -= 1
        dScores = (1 / N) * dScores

        # Return the gradient
        return dScores
```

- Đào tạo Neural:

```
import numpy as np


def train(network, X_train, y_train, X_val, y_val,
          batch_size, epochs, report_after=10, decay_after=50,
          learning_rate = 1e-3, learning_rate_decay=0.99):
    """
    Train a neural network
    Parameters:
```

```
    - network: the neural network to train
    - X_train: the train features of size (N x D)
    - y_train: the train labels of size (N,)
    - X_val: the validation features of size (N x D)
    - y_val: the validation labels of size (N,)
    - batch_size: the size of one training batch
    - epochs: the number of epochs
    Returns:
    - The costs of each epochs
    """
    # Initialize the costs array
    training_costs = []
    val_costs = []

    for epoch in range(epochs):
        # Epoch cost
        epoch_costs = []

        # Shuffle the training set
        random_indices = np.random.choice(X_train.shape[0], size=X_train.shape[0],
replace=False)
        X_shuffled = X_train[random_indices]
        y_shuffled = y_train[random_indices]

        # Get the batches
        X_batches = np.array_split(X_shuffled, batch_size)
        y_batches = np.array_split(y_shuffled, batch_size)

        # Forward pass and Backward pass
        for X_batch, y_batch in zip(X_batches, y_batches):
            # Perform the forward pass
            cost = network.forward(X_batch, y_batch, training=True)
            # Record the cosst
            epoch_costs.append(cost)

        # Calculate the average training cost
        avg_epoch_cost = np.mean(epoch_costs)
        training_costs.append(avg_epoch_cost)

        # Calculate the validation cost
        val_cost = network.forward(X_val, y_val, training=False)
        val_costs.append(val_cost)

        # Display the cost
        if (epoch + 1) % report_after == 0:
```

```
            # Print the train and validation accuracy
            y_train_predicted = network.predict(X_train)
            train_acc = (y_train_predicted == y_train).mean()
            y_val_predicted = network.predict(X_val)
            val_acc = (y_val_predicted == y_val).mean()
            print("------------------------------------------------------------------------")
            print(f"Epoch {epoch}: average cost {avg_epoch_cost:.2f}, train accuracy
{train_acc:.2f}, val accuracy {val_acc:.2f}.")

        # Learning rate update
        if (epoch + 1) % decay_after == 0:
            learning_rate = learning_rate * learning_rate_decay
            network.update_learning_rate(learning_rate)

    return training_costs, val_costs
```

- Mạng Neural:

```
import numpy as np
from layers import Linear, LinearAndReLU, Softmax
import pickle


"""
Define the neural network class
"""
class NeuralNetwork:
    def __init__(self, input_dim, hidden_dims, output_dim, regularization,
learning_rate):
        """
        Initialize the layers
        """
        # Initialize linear and relu layer
        dimensions = [input_dim] + hidden_dims
        self.layers = []
        for i in range(1, len(dimensions)):
            self.layers.append(LinearAndReLU(dimensions[i-1], dimensions[i],
regularization, learning_rate))

        # Initialize the final layer
        self.final_layer = Linear(output_dim, dimensions[-1], regularization,
learning_rate)

        # Initialize Softmax loss
        self.softmax = Softmax()
```

```
def update_learning_rate(self, learning_rate):
    """
    Update the learning rates of the layers
    """
    for layer in self.layers:
        layer.linear.lr = learning_rate
    self.final_layer.lr = learning_rate

def predict(self, X):
    """
    Perform the prediction of the Neural Network
    """
    # Initialize the current value
    hidden = X

    # Pass through the layers
    for layer in self.layers:
        hidden = layer.forward(hidden)

    # Pass through the final layer
    scores = self.final_layer.forward(hidden)

    return np.argmax(scores, axis=1)

def forward(self, X, y, training=True):
    """
    Perform the forward pass of the neural network
    """
    # Initialize the current value
    hidden = X

    # Pass through the layers
    for layer in self.layers:
        hidden = layer.forward(hidden)

    # Pass through the final layer
    scores = self.final_layer.forward(hidden)

    # Calculate the cost
    cost = self.softmax.forward(scores, y)

    if not training:
        return cost
```

```
        # Perform backward pass on final layers
        dLoss = self.softmax.backward(y)
        dLoss, _, _ = self.final_layer.backward(dLoss)

        # Perform backward pass on the layers
        for i in range(len(self.layers) - 1, -1, -1):
            dLoss, _, _ = self.layers[i].backward(dLoss)

        # Return the cost
        return cost

    @staticmethod
    def save(model):
        """
        Save the neural network into a pickle file
        """
        with open("model/model.pickle", "wb") as f:
            pickle.dump(model, f, protocol=pickle.HIGHEST_PROTOCOL)

    @staticmethod
    def load():
        """
        Load the model from a pickle file
        """
        with open("model/model.pickle", "rb") as f:
            model = pickle.load(f)
        return model
```

- Hàm chính:

```
import matplotlib.pyplot as plt
import numpy as np
from data_loader import scaler, X_train, y_train, X_val, y_val, X_test, y_test
from neural_net import NeuralNetwork
from train import train


# Create neural network
net = NeuralNetwork(
    input_dim=784,
    hidden_dims=[10],
    output_dim=10,
    regularization=5e-6,
    learning_rate=1e-3
)
```

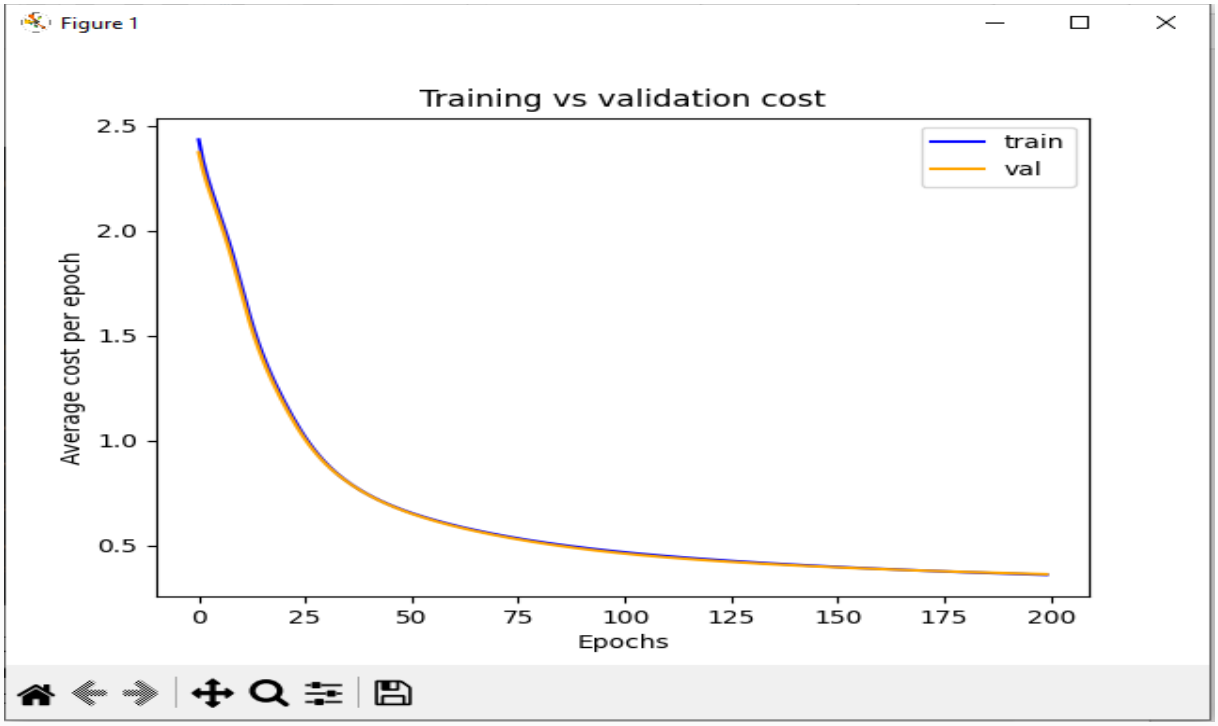
```
# Train the network
training_costs, val_costs = train(net, X_train, y_train, X_val, y_val, batch_size=200,
epochs=200)

# Check the test accuracy
y_test_predicted = net.predict(X_test)
test_acc = (y_test_predicted == y_test).mean()
print(f"Test accuracy: {test_acc:.2f}.")

# Save model
NeuralNetwork.save(net)

# Render the cost
fig, ax = plt.subplots()
ax.plot(training_costs, color="blue", label="train")
ax.plot(val_costs, color="orange", label="val")
ax.set_xlabel("Epochs")
ax.set_ylabel("Average cost per epoch")
ax.set_title("Training vs validation cost")
plt.legend()
plt.show()

# Display the correctly classified examples
correct_examples = X_test[y_test == y_test_predicted]
correctly_classified =
scaler.inverse_transform(correct_examples[np.random.randint(correct_examples.shape
[0], size=4), :])
plt.subplot(221)
plt.imshow(correctly_classified[0].reshape(28, 28), cmap='gray')
plt.subplot(222)
plt.imshow(correctly_classified[1].reshape(28, 28), cmap='gray')
plt.subplot(223)
plt.imshow(correctly_classified[2].reshape(28, 28), cmap='gray')
plt.subplot(224)
plt.imshow(correctly_classified[3].reshape(28, 28), cmap='gray')
plt.show()

# Display the incorrect classified examples
incorrect_examples = X_test[y_test != y_test_predicted]
incorrectly_classified =
scaler.inverse_transform(incorrect_examples[np.random.randint(incorrect_examples.s
hape[0], size=4), :])
plt.subplot(221)
plt.imshow(incorrectly_classified[0].reshape(28, 28), cmap='gray')
```

```
plt.subplot(222)
plt.imshow(incorrectly_classified[1].reshape(28, 28), cmap='gray')
plt.subplot(223)
plt.imshow(incorrectly_classified[2].reshape(28, 28), cmap='gray')
plt.subplot(224)
plt.imshow(incorrectly_classified[3].reshape(28, 28), cmap='gray')
plt.show()
```

- Kết quả:

```
Epoch 9: average cost 1.82, train accuracy 0.43, val accuracy 0.43.
--------------------------------------------------------------------------------
Epoch 19: average cost 1.23, train accuracy 0.60, val accuracy 0.59.
--------------------------------------------------------------------------------
Epoch 29: average cost 0.91, train accuracy 0.72, val accuracy 0.71.
--------------------------------------------------------------------------------
Epoch 39: average cost 0.75, train accuracy 0.76, val accuracy 0.76.
--------------------------------------------------------------------------------
Epoch 49: average cost 0.66, train accuracy 0.79, val accuracy 0.79.
--------------------------------------------------------------------------------
Epoch 59: average cost 0.60, train accuracy 0.81, val accuracy 0.81.
--------------------------------------------------------------------------------
Epoch 69: average cost 0.56, train accuracy 0.83, val accuracy 0.83.
--------------------------------------------------------------------------------
Epoch 79: average cost 0.52, train accuracy 0.84, val accuracy 0.84.
--------------------------------------------------------------------------------
Epoch 89: average cost 0.49, train accuracy 0.85, val accuracy 0.85.
--------------------------------------------------------------------------------
Epoch 99: average cost 0.47, train accuracy 0.86, val accuracy 0.85.
--------------------------------------------------------------------------------
```

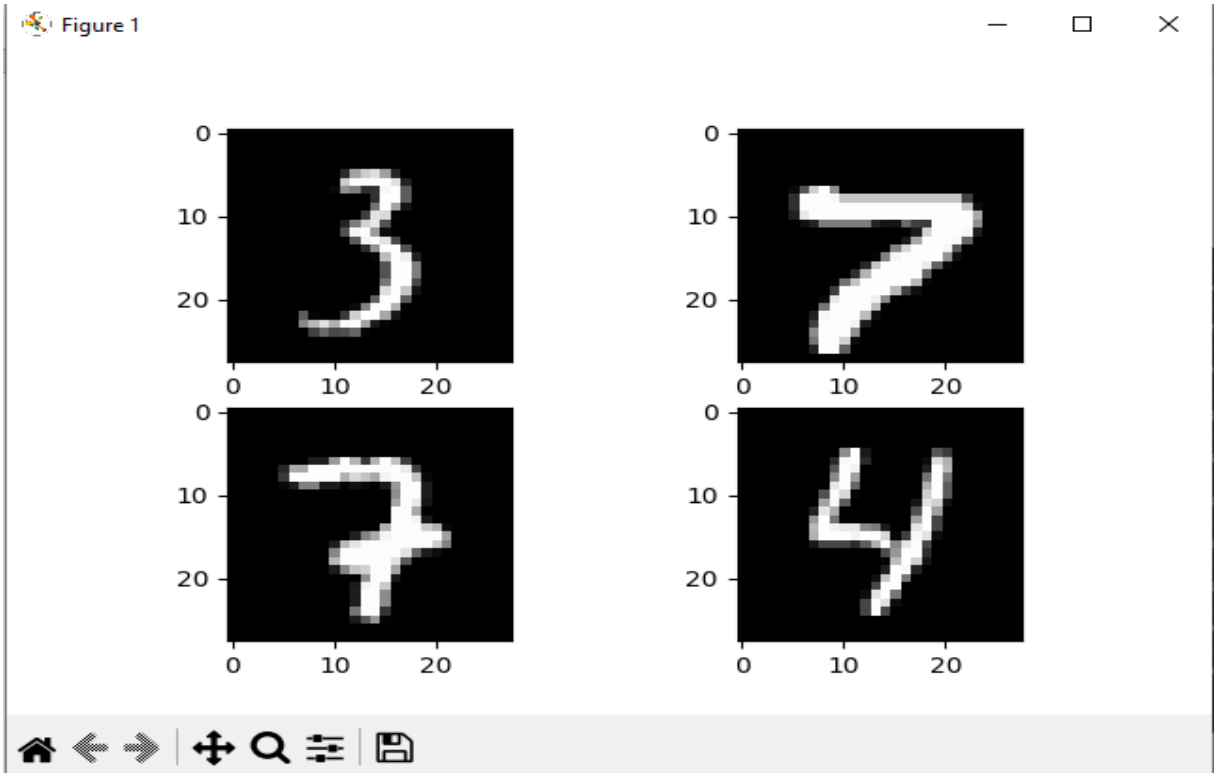
```
--------------------------------------------------------------------------------
Epoch 109: average cost 0.45, train accuracy 0.86, val accuracy 0.86.
--------------------------------------------------------------------------------
Epoch 119: average cost 0.43, train accuracy 0.87, val accuracy 0.87.
--------------------------------------------------------------------------------
Epoch 129: average cost 0.42, train accuracy 0.87, val accuracy 0.87.
--------------------------------------------------------------------------------
Epoch 139: average cost 0.41, train accuracy 0.88, val accuracy 0.88.
--------------------------------------------------------------------------------
Epoch 149: average cost 0.40, train accuracy 0.88, val accuracy 0.88.
--------------------------------------------------------------------------------
Epoch 159: average cost 0.39, train accuracy 0.88, val accuracy 0.88.
--------------------------------------------------------------------------------
Epoch 169: average cost 0.38, train accuracy 0.89, val accuracy 0.88.
--------------------------------------------------------------------------------
Epoch 179: average cost 0.37, train accuracy 0.89, val accuracy 0.88.
--------------------------------------------------------------------------------
Epoch 189: average cost 0.37, train accuracy 0.89, val accuracy 0.89.
--------------------------------------------------------------------------------
Epoch 199: average cost 0.36, train accuracy 0.89, val accuracy 0.89.
Test accuracy: 0.89.
```



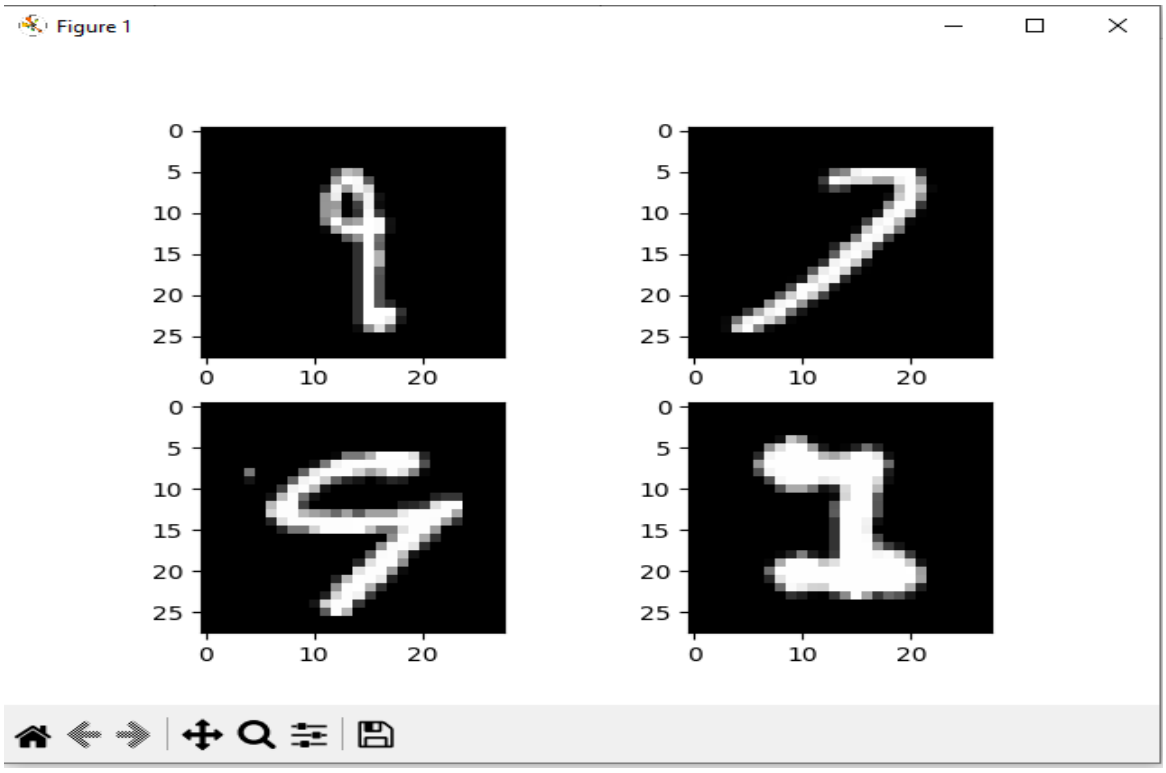
- Giá trị của hàm lost giảm dần theo thời gian trong quá trình huấn luyện: Từ 1.72 xuống còn 0.36

- Độ chính xác của mạng nơ-ron nhân tạo đối với Loss function và Validation cũng tăng dần theo thời gian: Từ 0.43 lên 0.89.

- Những chữ số viết tay mà mạng nơ-ron đoán đúng:



- Những chữ số mà mạng nơ-ron nhân tạo dự đoán sai:



III. Phân lớp với Naïve Bayes

1.Khái niệm.

Một phân loại Naive Bayes dựa trên ý tưởng nó là một lớp được dự đoán bằng các giá trị của đặc trưng cho các thành viên của lớp đó. Các đối tượng là một nhóm (group) trong các lớp nếu chúng có cùng các đặc trưng chung. Có thể có nhiều lớp rời rạc hoặc lớp nhị phân.

Các luật Bayes dựa trên xác suất để dự đoán chúng về các lớp có sẵn dựa trên các đặc trưng được trích rút. Trong phân loại Bayes, việc học được coi như xây dựng một mô hình xác suất của các đặc trưng và sử dụng mô hình này để dự đoán phân loại cho một ví dụ mới.

Biến chưa biết hay còn gọi là biến ẩn là một biến xác suất chưa được quan sát trước đó. Phân loại Bayes sử dụng mô hình xác suất trong đó phân loại là một biến ẩn có liên quan tới các biến đã được quan sát. Quá trình phân loại lúc này trở thành suy diễn trên mô hình xác suất.

Trường hợp đơn giản nhất của phân loại Naive Bayes là tạo ra các giả thiết độc lập về các đặc trưng đầu vào và độc lập có điều kiện với mỗi một lớp đã cho. Sự độc lập của phân loại Naive Bayes chính là thể hiện của mô hình mạng tin cậy (belief network) trong trường hợp đặc biệt, và phân loại là chỉ dựa trên một nút cha duy nhất của mỗi một đặc trưng đầu vào. Mạng tin cậy này đề cập tới xác suất phân tán P(Y) đối với mỗi một đặc trưng đích Y và P(Xi |Y) đối với mỗi một đặc trưng đầu vào Xi . Với mỗi một đối tượng, dự đoán bằng cách tính toán dựa trên các xác suất điều kiện của các đặc trưng quan sát được cho mỗi đặc trưng đầu vào.

2. Định lý Bayes.

Gọi A, B là hai biến cố

- P(A): Xác suất của sự kiện A xảy ra.
- P(B): Xác suất của sự kiện B xảy ra.
- P(B|A): Xác suất (có điều kiện) của sự kiện B xảy ra, nếu biết rằng sự kiện A đã xảy ra.
- P(A|B): Xác suất (có điều kiện) của sự kiện A xảy ra, nếu biết rằng sự kiện B đã xảy ra.

Với P(B) > 0:

$$P(A|B) = \frac{P(AB)}{P(B)}$$

Suy ra:

$$P(AB) = P(A|B)P(B) = P(B|A)P(A)$$

Công thức Bayes:

$$P(B|A) = \frac{P(AB)}{P(A)} = \frac{P(A|B)P(B)}{P(A)} = \frac{P(A|B)P(B)}{P(AB) + P(A\overline{B})}$$

$$= \frac{P(A|B)P(B)}{P(AB) + P(A\overline{B})} = \frac{P(A|B)P(B)}{P(A|B)P(B) + P(A|\overline{B})P(\overline{B})}$$

- Công thức Bayes tổng quát:

Với P(A) > 0 và $\{B_1, B_2, ..., B_n\}$ là một hệ đầy đủ các biến cố:

- Tổng xác suất của hệ bằng 1:

$$\sum_{k=1}^{n} P(B_k) = 1$$

- Từng đôi một xung khắc:

$$P(B_i \cap B_j) = 0$$

Khi đó ta có:

$$P(B_k | A) = \frac{P(A|B_k)P(B_k)}{P(A)}$$

$$= \frac{P(A|B_k)P(B_k)}{\sum_{i=1}^{n} P(A|B_i)P(B_i)}$$

Trong đó ta gọi A là một chứng cứ (evidence) (trong bài toán phân lớp A sẽ là một phần tử dữ liệu), B là một giả thiết nào để cho A thuộc về một lớp C nào đó. Trong bài toán phân lớp chúng ta muốn xác định giá trị P(B/A) là xác suất để giả thiết B là đúng với chứng cứ A thuộc vào lớp C với điều kiện ra đã biết các thông tin mô tả A. P(B|A) là một xác suất hậu nghiệm (posterior probability hay posteriori probability) của B với điều kiện A.

Giả sử tập dữ liệu liệu khách hàng của chúng ta được mô tả bởi các thuộc tính tuổi và thu nhập, và một khách hàng X có tuổi là 25 và thu nhập là 2000$. Giả sử H là

giả thiết khách hàng đõ sẽ mua máy tính, thì P(H|X) phản ánh xác xuất người dùng X sẽ mua máy tính với điều kiện ta biết tuổi và thu nhập của người đó.

Ngược lại P(H) là xác suất tiền nghiệm (prior probability hay priori probability) của H. Trong ví dụ trên, nó là xác suất một khách hàng sẽ mua máy tính mà không cần biết các thông tin về tuổi hay thu nhập của họ. Hay nói cách khác, xác suất này không phụ thuộc vào yếu tố X. Tương tự, P(X|H) là xác suất của X với điều kiện H (likelihood), nó là một xác suất hậu nghiệm.

Ví dụ, nó là xác suất người dùng X (có tuổi là 25 và thu nhập là $200) sẽ mua máy tính với điều kiện ta đã biết người đó sẽ mua máy tính. Cuối cùng P(X) là xác suất tiền nghiệm của X. Trong ví dụ trên, nó se là xác xuất một người trong tập dữ liệu sẽ có tuổi 25 và thu nhập $2000.

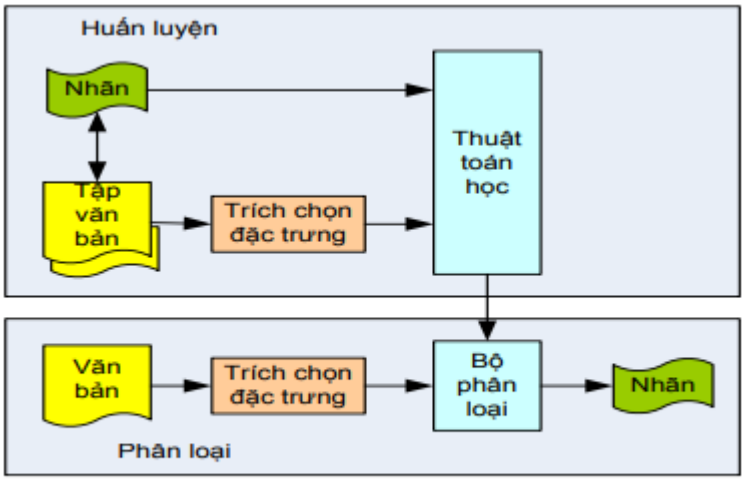
Posterior = Likelihood * Prior / Evidence.

3. Phân lớp Naive Bayes.

Naive Bayes là phương pháp phân loại dựa vào xác suất được sử dụng rộng rãi trong lĩnh vực máy học và nhiều lĩnh vực khác như trong các công cụ tìm kiếm, các bộ lọc mail.

Bộ phân lớp Naive bayes hay bộ phân lớp Bayes (simple byes classifier) hoạt động như sau:

- Gọi D là tập dữ liệu huấn luyện, trong đó mỗi phần tử dữ liệu X được biểu diễn bằng một vector chứa n giá trị thuộc tính A1, A2,...,An = {x1,x2,...,xn}
- Giả sử có m lớp C1, C2,..,Cm. Cho một phần tử dữ liệu X, bộ phân lớp sẽ gán nhãn cho X là lớp có xác suất hậu nghiệm lớn nhất. Cụ thể, bộ phân lớp Bayes sẽ dự đoán X thuộc vào lớp Ci nếu và chỉ nếu:
  P(Ci|X) > P(Cj|X) (1<= i, j <=m, i != j)
  Giá trị này sẽ tính dựa trên định lý Bayes.
- Để tìm xác suất lớn nhất, ta nhận thấy các giá trị P(X) là giống nhau với mọi lớp nên không cần tính. Do đó ta chỉ cần tìm giá trị lớn nhất của P(X|Ci) * P(Ci). Chú ý rằng P(Ci) được ước lượng bằng |Di|/|D|, trong đó Di là tập các phần tử dữ liệu thuộc lớp Ci. Nếu xác suất tiền nghiệm P(Ci) cũng không xác định được thì ta coi chúng bằng nhau P(C1) = P(C2) = ... = P(Cm), khi đó ta chỉ cần tìm giá trị P(X|Ci) lớn nhất.
- Khi số lượng các thuộc tính mô tả dữ liệu là lớn thì chi phí tính toàn P(X|Ci) là rất lớn, dó đó có thể giảm độ phức tạp của thuật toán Naive Bayes giả thiết các thuộc tính độc lập nhau. Khi đó ta có thể tính:
  P(X|Ci) = P(x1|Ci)...P(xn|Ci)



Hình 1. Mô tả bước xây dựng bộ phân lớp

4. Khắc phục vấn đề xác suất điều kiện bằng zero

- Nếu trong dữ liệu huấn luyện không có đối tượng X nào có thuộc tính lớp Ck có thuộc tính Fi nhận một giá trị cụ thể vij, xác suất điều kiện P(Fi = xij | Ck) sẽ bằng 0.
- Khi phân lớp, nếu có một đối tượng nào mang thuộc tính này thì xác suất phân vào lớp Ck luôn bằng 0.
- Khắc phục bằng cách ước lượng theo công thức sau:

$$P(F_i = v_j^i \mid C = c_k) = \frac{n_{ijk} + mp}{n_k + m}$$

Trong đó:

- n_{ijk} là số đối tượng **x** thuộc lớp c_k mang thuộc tính-giá trị (F_i, v_j^i) trong **D**.
- n_k là số lượng đối tượng **x** thuộc lớp c_k.
- p là một hằng số (p có thể = 1 hoặc = $1/P_i$ với P_i là số giá trị thuộc tính F_i có thể nhận).
- m là số lượng đối tượng x "ảo", m ≥ 1

5. Nhược điểm

- Giả định độc lập (ưu điểm cũng chính là nhược điểm)
  hầu hết các trường hợp thực tế trong đó có các thuộc tính trong các đối tượng thường phụ thuộc lẫn nhau.
- Vấn đề zero (đã nêu cách giải quyết ở phía trên)
- Mô hình không được huấn luyện bằng phượng pháp tối ưu mạnh và chặt chẽ.
  Tham số mủa mô hình là các ước lượng xác suất điều kiện đơn lẻ.
  Không tính đến sự tương tác giữa các ước lượng này.

6. Ứng dụng Naive Bayes trong phân loại tài liệu tiếng Việt.

- Đặc điểm.

Trong tất cả các ngôn ngữ, người ta thường phân chia dòng ngữ lưu thành các âm tiết. Âm tiết là đơn vị phát âm tối thiểu của lời nói. Nghiên cứu âm tiết tức là nghiên cứu sự tổ hợp các âm vị (phômen) trong dòng lưu ngữ, ví dụ như các thực từ.

Một điểm cơ bản nhất của các âm tiết tiếng Việt là ranh giới của âm tiết tiếng Việt trùng với ranh giới của hình vị (moocphem), tức là mỗi âm tiết đều đóng vai trò là dấu hiệu của một hình vị (moocphem), đơn vị có nghĩa dùng làm thành tố cấu tạo từ. Lời nói của con người là một chuỗi âm thanh được phát ra kế tiếp nhau trong không gian và thời gian. Việc phân tích chuỗi âm thanh ấy người ta nhận ra được các đơn vị của ngữ âm.

Đặc điểm thứ hai của âm tiết tiếng Việt là mỗi âm tiết tiếng Việt đều gắn liền với một trong sáu thanh điệu (không, huyền, ngã, hỏi, sắc, nặng) vì tiếng Việt là loại ngôn ngữ có thanh điệu khác với ngôn ngữ khác. Thanh điệu tham gia vào việc cấu tạo từ, làm chức năng phân biệt ý nghĩa của từ và làm dấu hiệu phân biệt từ. Thanh điệu có chức năng như một âm vị, nó gắn liền với âm tiết và biểu hiện trong toàn âm tiết [2].

Do đặc điểm trên mà âm tiết có vị trí rất quan trọng trong việc nghiên cứu âm tiếng Việt. Muốn xác định thành phần âm vị của ngôn ngữ, người ta thường xuất phát từ việc xác định các hình vị rồi từ các moocphem đó mà phân tích ra các âm vị, hình vị trong tiếng Việt trùng hợp với các âm tiết; chúng ta xuất phát từ việc phân tích các âm tiết để xác định các âm vị. Nếu như trong ngôn ngữ Ấn – Âu, âm tiết chỉ là vấn đề thuộc hàng thứ yếu so với âm vị và hình vị thì trong tiếng Việt, âm tiết là vấn đề hàng đầu của âm vị học.

- Cấu trúc âm tiết

Mỗi âm tiết tiếng Việt là một khối hoàn chỉnh trong phát âm. Trong ngữ cảm của người Việt, âm tiết tuy được phát âm liền một hơi, nhưng không phải là một khối bất biến mà có cấu tạo lắp ghép. Khối lắp ghép ấy có thể tháo rời từng bộ phận của âm tiết này để hoán vị với bộ phận tương ứng ở âm tiết khác. Mỗi âm tiết tiếng Việt có 3 bộ phận: phụ âm đầu, vần và thanh điệu.

- Giảm chiều đặc trưng bằng mô hình chủ đề.

Trong học máy và xử lý ngôn ngữ tự nhiên, một mô hình chủ đề là một loại mô hình thống kê để phát hiện ra các "chủ đề" trừu tượng xảy ra trong một bộ sưu tập các tài liệu. Một số phương pháp xây dựng mô hình chủ đề như: Xây dựng mô hình chủ đề dựa trên phân phối ẩn Dirichlet; Mô hình dựa trên mạng Bayesian; Mô hình chủ đề xây dựng dựa trên mô hình Markov ẩn.

- Xây dựng mô hình chủ đề cho tiếng Việt.

Xử lý giảm số chiều của đặc trưng bằng cách sử dụng mô hình chủ đề, do đó số lượng thuật ngữ trong mỗi văn bản sẽ giảm hơn nhiều so với số các từ trong một văn bản, mặt khác sẽ giải quyết bài toán tách từ tiếng Việt nhờ đó làm tăng độ chính xác của hệ thống, tiếp theo áp dụng lý thuyết Naive Bayes để phân loại các văn bản theo đúng chủ đề đã chọn [11].

- Phân loại văn bản tiếng Việt dựa trên Naive Bayes.

Sau khi xây dựng được tập từ chủ đề đối với mỗi một lớp chủ đề. Tiếp theo sử dụng phân loại Naive Bayes để xây dựng mô hình phân loại tự động.

Sử dụng luật cực đại hóa hậu nghiệm (Maximum a posterioriMAP) có công thức sau:

$$c_{map} = \arg\max_{c \in C}(P(c|d)) = \arg\max_{c \in C}\left(P(c)\prod_{1 \le k \le n_d} P(t_k|c)\right) \quad (1)$$

Trong đó:

- T_k: các từ của tài liệu;
- C: chủ đề;
- $P(c|d)$: xác suất điều kiện của lớp c với tài liệu đã cho d;
- $P(c)$: xác suất tiền nghiệm của lớp c;
- $P(t_k|c)$: xác suất điều kiện của từ T_k với lớp c đã cho.

Sử dụng luật biến đổi Laplace cho công thức (1) chuyển thành

$$P(t|c) = \frac{T_{ct}+1}{\sum_{t' \in V}(T_{ct'}+1)} = \frac{T_{ct}+1}{\sum_{t' \in V}(T_{ct'}) + B'} \quad (2)$$

Trong đó B’ là tổng số tất cả các từ chủ đề, Tct là số lần xuất hiện của thuật ngữ t trong các tài liệu huấn luyện thuộc lớp c.

7. Xây dựng Phân loại Naïve Bayes.

```
import numpy as np


class NaiveBayes:
    def fit(self, X, y):
        n_samples, n_features = X.shape
        self._classes = np.unique(y)
        n_classes = len(self._classes)

        # calculate mean, var, and prior for each class
        self._mean = np.zeros((n_classes, n_features), dtype=np.float64)
        self._var = np.zeros((n_classes, n_features), dtype=np.float64)
        self._priors = np.zeros(n_classes, dtype=np.float64)

        for idx, c in enumerate(self._classes):
            X_c = X[y == c]
            self._mean[idx, :] = X_c.mean(axis=0)
            self._var[idx, :] = X_c.var(axis=0)
            self._priors[idx] = X_c.shape[0] / float(n_samples)

    def predict(self, X):
        y_pred = [self._predict(x) for x in X]
        return np.array(y_pred)

    def _predict(self, x):
        posteriors = []

        # calculate posterior probability for each class
        for idx, c in enumerate(self._classes):
            prior = np.log(self._priors[idx])
            posterior = np.sum(np.log(self._pdf(idx, x)))
            posterior = prior + posterior
            posteriors.append(posterior)

        # return class with highest posterior probability
        return self._classes[np.argmax(posteriors)]

    def _pdf(self, class_idx, x):
        mean = self._mean[class_idx]
        var = self._var[class_idx]
        numerator = np.exp(-((x - mean) ** 2) / (2 * var))
        denominator = np.sqrt(2 * np.pi * var)
        return numerator / denominator
```

```
# Testing
if __name__ == "__main__":
    # Imports
    from sklearn.model_selection import train_test_split
    from sklearn import datasets

    def accuracy(y_true, y_pred):
        accuracy = np.sum(y_true == y_pred) / len(y_true)
        return accuracy

    X, y = datasets.make_classification(
        n_samples=1000, n_features=10, n_classes=2, random_state=123
    )
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=123
    )

    nb = NaiveBayes()
    nb.fit(X_train, y_train)
    predictions = nb.predict(X_test)

    print("Naive Bayes classification accuracy", accuracy(y_test, predictions))
```

- Kết quả sau khi chạy chương trình:

```
Naive Bayes classification accuracy 0.965
```

Tài liệu tham khảo:

Tài liệu tiếng Việt

1. Nguyễn Linh Giang, Nguyễn Mạnh Hiển, Phân loại văn bản tiếng Việt với

bộ phân loại vectơ hỗ trợ SVM, 2002.

2. Nguyễn Hữu Quỳnh, Ngữ pháp Tiếng Việt, NXB Từ điển Bách Khoa, 2001.

Amazon Web Services. (n.d.). Mạng nơ-ron là gì? From
https://aws.amazon.com/vi/what-is/neural-network/

IBM. (n.d.). What is a neural network? from https://www.ibm.com/topics/neural-networks

IBM. (n.d.). What is supervised learning? From https://www.ibm.com/topics/supervised-learning

NTTuan8. (2019, March 9). Bài 3: Neural network. From https://nttuan8.com/bai-3-neural-network/