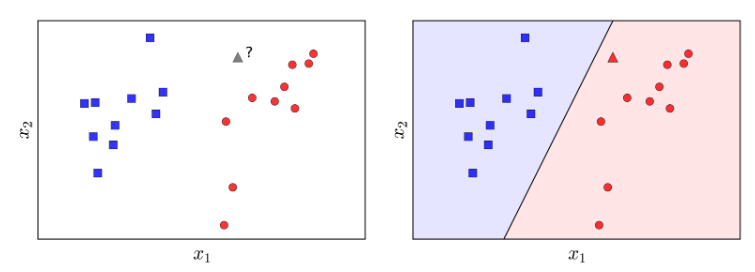
**PERCEPTRON**

1. **Giới thiệu bài toán Perceptron**

Perceptron Learning Algorithm (PLA) hoặc đôi khi được viết gọn là Perceptron  là một thuật toán Classification cho trường hợp đơn giản nhất: chỉ có hai class (lớp) (bài toán với chỉ hai class được gọi là binary classification) và cũng chỉ hoạt động được trong một trường hợp rất cụ thể. Tuy nhiên, nó là nền tảng cho một mảng lớn quan trọng của Machine Learning là Neural Networks và sau này là Deep Learning.

Giả sử chúng ta có hai tập hợp dữ liệu đã được gán nhãn được minh hoạ trong Hình 1 bên trái dưới đây. Hai class của chúng ta là tập các điểm màu xanh và tập các điểm màu đỏ. Bài toán đặt ra là: từ dữ liệu của hai tập được gán nhãn cho trước, hãy xây dựng một classifier (bộ phân lớp) để khi có một điểm dữ liệu hình tam giác màu xám mới, ta có thể dự đoán được màu (nhãn) của nó. Điều này có nghĩa là phải tìm ra một ranh giới ( có thể là đường thẳng, mặt phẳng, ..) để phân chia 2 tập hợp với nhau. Đây cũng chính là bài toán Perceptron!



* **Bài toán Perceptron**

Bài toán Perceptron được phát biểu như sau: *Cho hai class được gán nhãn, hãy tìm một đường phẳng sao cho toàn bộ các điểm thuộc class 1 nằm về 1 phía, toàn bộ các điểm thuộc class 2 nằm về phía còn lại của đường phẳng đó. Với giả định rằng tồn tại một đường phẳng như thế.*

**2. Thuật toán Perceptron (PLA)**

Ý tưởng cơ bản của PLA là xuất phát từ một nghiệm dự đoán nào đó, qua mỗi vòng lặp, nghiệm sẽ được cập nhật tới một ví trí tốt hơn. Việc cập nhật này dựa trên việc giảm giá trị của một hàm mất mát nào đó.

Giả sử **X=[x1,x2,…,xN]∈Rd×N** là ma trận chứa các điểm dữ liệu mà mỗi cột xi∈Rd×1 là một điểm dữ liệu trong không gian d chiều một hàm mất mát nào đó.

Giả sử thêm các nhãn tương ứng với từng điểm dữ liệu được lưu trong một vector hàng **y=[y1,y2,…,yN]∈R1×N**, với yi=1 nếu xi thuộc class 1 (xanh) và yi=−1 nếu xi thuộc class 2 (đỏ).

Tại một thời điểm, giả sử ta tìm được boundary là đường phẳng có phương trình:

**fw(x)=w1x1+⋯+wdxd+w0 = wT \* X=0**

với X là điểm dữ liệu mở rộng bằng cách thêm phần tử x0=1 lên trước vector x tương tự như trong[*Linear Regression*](https://machinelearningcoban.com/2016/12/28/linearregression/). Và từ đây, khi nói x, ta cũng ngầm hiểu là điểm dữ liệu mở rộng.

Nhận xét rằng các điểm nằm về cùng 1 phía so với đường thẳng này sẽ làm cho hàm số fw(x) mang cùng dấu. Chỉ cần đổi dấu của w nếu cần thiết, ta có thể giả sử các điểm nằm trong nửa mặt phẳng nền xanh mang dấu dương (+), các điểm nằm trong nửa mặt phẳng nền đỏ mang dấu âm (-). Các dấu này cũng tương đương với nhãn y của mỗi class. Vậy nếu w là một nghiệm của bài toán Perceptron, với một điểm dữ liệu mới x chưa được gán nhãn, ta có thể xác định class của nó bằng phép toán đơn giản như sau:

**label(x)=1 if wTx≥0** **và ngược lại**

Ngắn gọn hơn:

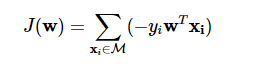
**label(x)=sgn(wTx)**

trong đó, sgn là hàm xác định dấu, với giả sử rằng sgn(0)=1.

1. **Hàm mất mát**

Tại một thời điểm, nếu chúng ta chỉ quan tâm tới các điểm bị misclassified thì hàm số J(w) khả vi (tính được đạo hàm), vậy chúng ta có thể sử dụng [Gradient Descent](https://machinelearningcoban.com/2017/01/12/gradientdescent/) hoặc [Stochastic Gradient Descent (SGD)](https://machinelearningcoban.com/2017/01/16/gradientdescent2/#-stochastic-gradient-descent) để tối ưu hàm mất mát này. Với ưu điểm của SGD cho các bài toán [large-scale](https://machinelearningcoban.com/2017/01/12/gradientdescent/#large-scale), chúng ta sẽ làm theo thuật toán này.

Với *một* điểm dữ liệu xi bị misclassified, hàm mất mát trở thành:

****

Đạo hàm tương ứng:

****

Vậy quy tắc cập nhật là:



với η là learning rate được chọn bằng 1. Ta có một quy tắc cập nhật rất gọn là: wt+1=wt+yixi. Nói cách khác, với mỗi điểm xi bị misclassifed, ta chỉ cần nhân điểm đó với nhãn yi của nó, lấy kết quả cộng vào w ta sẽ được w mới.

**Ưu điểm** của hàm mất mát này làkhi một điểm misclassified xi nằm càng xa boundary thì giá trị −yiwTxi sẽ càng lớn, nghĩa là sự sai lệch càng lớn. Giá trị nhỏ nhất của hàm mất mát này cũng bằng 0 nếu không có điểm nào bị misclassifed. Hàm mất mát này trừng phạt rất nặng những điểm lấn sâu sang lãnh thổ của class kia.

**4. Tóm tắt PLA**

1. Chọn ngẫu nhiên một vector hệ số w với các phần tử gần 0.
2. Duyệt ngẫu nhiên qua từng điểm dữ liệu xi:
   * Nếu xi được phân lớp đúng, tức **sgn(wTxi)=yi**, chúng ta không cần làm gì.
   * Nếu xi bị misclassifed, cập nhật w theo công thức:

**w=w+yixi**

1. Kiểm tra xem có bao nhiêu điểm bị misclassifed. Nếu không còn điểm nào, dừng thuật toán. Nếu còn, quay lại bước 2.

**5. Code Implement:**

# Perceptron Algorithm

from random import seed

from random import randrange

from csv import reader

# Load a CSV file

def load\_csv(filename):

dataset = list()

with open(filename, 'r') as file:

csv\_reader = reader(file)

for row in csv\_reader:

if not row:

continue

dataset.append(row)

return dataset

# Chuyển đổi cột chuỗi thành float

def str\_column\_to\_float(dataset, column):

for row in dataset:

row[column] = float(row[column].strip())

# Chuyển đổi cột chuỗi thành số nguyên

def str\_column\_to\_int(dataset, column):

class\_values = [row[column] for row in dataset]

unique = set(class\_values)

lookup = dict()

for i, value in enumerate(unique):

lookup[value] = i

for row in dataset:

row[column] = lookup[row[column]]

return lookup

# Chia tập dữ liệu thành k nếp gấp

def cross\_validation\_split(dataset, n\_folds):

dataset\_split = list()

dataset\_copy = list(dataset)

fold\_size = int(len(dataset) / n\_folds)

for i in range(n\_folds):

fold = list()

while len(fold) < fold\_size:

index = randrange(len(dataset\_copy))

fold.append(dataset\_copy.pop(index))

dataset\_split.append(fold)

return dataset\_split

# Tính phần trăm độ chính xác

def accuracy\_metric(actual, predicted):

correct = 0

for i in range(len(actual)):

if actual[i] == predicted[i]:

correct += 1

return correct / float(len(actual)) \* 100.0

# Đánh giá thuật toán bằng cách sử dụng phân tách xác thực chéo

def evaluate\_algorithm(dataset, algorithm, n\_folds, \*args):

folds = cross\_validation\_split(dataset, n\_folds)

scores = list()

for fold in folds:

train\_set = list(folds)

train\_set.remove(fold)

train\_set = sum(train\_set, [])

test\_set = list()

for row in fold:

row\_copy = list(row)

test\_set.append(row\_copy)

row\_copy[-1] = None

predicted = algorithm(train\_set, test\_set, \*args)

actual = [row[-1] for row in fold]

accuracy = accuracy\_metric(actual, predicted)

scores.append(accuracy)

return scores

# Đưa ra dự đoán với trọng số

def predict(row, weights):

activation = weights[0]

for i in range(len(row)-1):

activation += weights[i + 1] \* row[i]

return 1.0 if activation >= 0.0 else 0.0

# Ước tính trọng số Perceptron bằng cách sử dụng gốc gradient ngẫu nhiên

def train\_weights(train, l\_rate, n\_epoch):

weights = [0.0 for i in range(len(train[0]))]

for epoch in range(n\_epoch):

for row in train:

prediction = predict(row, weights)

error = row[-1] - prediction

weights[0] = weights[0] + l\_rate \* error

for i in range(len(row)-1):

weights[i + 1] = weights[i + 1] + l\_rate \* error \* row[i]

return weights

# Thuật toán Perceptron với Stochastic Gradient Descent

def perceptron(train, test, l\_rate, n\_epoch):

predictions = list()

weights = train\_weights(train, l\_rate, n\_epoch)

for row in test:

prediction = predict(row, weights)

predictions.append(prediction)

return(predictions)

# Kiểm tra thuật toán Perceptron

seed(1)

# tải và chuẩn bị dữ liệu

filename = 'Sinhvien-1.csv'

dataset = load\_csv(filename)[1:]

for i in range(len(dataset[0])-1):

str\_column\_to\_float(dataset, i)

# chuyển đổi lớp chuỗi thành số nguyên

str\_column\_to\_int(dataset, len(dataset[0])-1)

# đánh giá thuật toán

n\_folds = 3

l\_rate = 0.01

n\_epoch = 500

scores = evaluate\_algorithm(dataset, perceptron, n\_folds, l\_rate, n\_epoch)

print('Scores: %s' % scores)

print('Mean Accuracy: %.3f%%' % (sum(scores)/float(len(scores))))