**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN**

**KHOA CÔNG NGHỆ THÔNG TIN**

**BỘ MÔN KHOA HỌC MÁY TÍNH**

- - - - 🙞🙞🟔🙜🙜 - - - -

**NHÓM LOTUS**

**0712228 – TRẦN TRUNG KIÊN**

**0712263 – VẠN DUY THANH LONG**

**PHÁT HIỆN KHUÔN MẶT VỚI THUẬT TOÁN ADABOOST**

**ĐỒ ÁN MÔN HỌC KỸ THUẬT TRÍ TUỆ NHÂN TẠO**

**GIẢNG VIÊN LÝ THUYẾT**

**TS. LÊ HOÀNG THÁI**

**NĂM 2010**

# MỤC LỤC

MỤC LỤC 2

# NỀN TẢNG LÝ THUYẾT

## Đặc trưng Haar-like

### Tổng quan về đặc trưng Haar-like

Mỗi đặc trưng Haar-like chứa 2, 3 hoặc 4 miền hình học có màu “trắng” hoặc “đen.” Để rõ hơn, ta hãy ngó qua các đặc trưng Haar-like dưới đây:

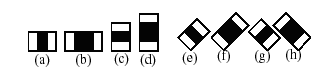
* Tập các đặc trưng Haar-like cơ sở:



* Tập các đặc trưng Haar-like mở rộng:
  + Đặc trưng cạnh:



* + Đặc trưng đường:



* + Đặc trưng tâm – xung quanh:



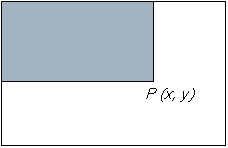
Giá trị của một đặc trưng Haar-like chỉ đơn giản là sự khác biệt giữa tổng các giá trị xám của các pixel trong vùng “đen” với tổng các giá trị xám của các pixel trong vùng “trắng”:

*f(x)=Sumblack rectangle (pixel gray level) – Sumwhite rectangle (pixel gray level)*

Nhận thấy ý nghĩa của đặc trưng Haar-like là nó thể hiện được tri thức về các đối tượng trong ảnh (thông qua việc thể hiện mối quan hệ giữa các bộ phận của đối tượng.)

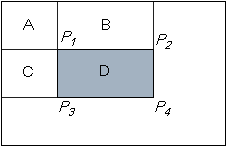
Ta tính giá trị của đặc trưng Haar-like rất nhanh thông qua một cách thức được gọi là “Integral Image.” Ở đây, Integral Image tại vị trí (x, y) có giá trị bằng tổng các giá trị pixel nằm trong hình chữ nhật được định bởi góc trái trên là (0, 0) và góc phải dưới là (x, y):





Từ đây, ta có thể tính được ngay tổng các giá trị pixel của một hình chữ nhật bất kỳ thông qua Integral Image tại 4 đỉnh của nó.

Ví dụ: Tính D.



Trong hình vẽ trên:

* A, B, C, D lần lượt là tổng các giá trị pixel của hình chữ nhật tương ứng.
* P1, P2, P3, P4 lần lượt là Integral Image tại 4 đỉnh của hình chữ nhật D.

Có ngay:

P1 = A, P2 = A + B, P3 = A + C, P4 = A + B + C + D

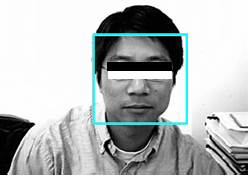
Suy ra:

P1 + P4 – P2 – P3 = A + (A + B + C + D) – (A + B) – (A + C) = D

Vậy ta có: **D = P1 + P4 – P2 – P3**

### Hình dung ban đầu về việc áp dụng đặc trưng Haar-like để phát hiện khuôn mặt

Để phát hiện khuôn mặt, ảnh được quét bằng một cửa sổ con chứa một đặc trưng Haar-like.



Ứng với một đặc trưng Haar-like fj, một bộ phân lớp yếu yj(x) được định nghĩa như sau:

Trong đó:

* x là cửa sổ con.
* là ngưỡng.
* là hệ số với ý nghĩa là quyết định dấu của bất phương trình.

Ta hiểu công thức này đơn giản như sau: nếu giá trị của đặc trưng Haar-like tại cửa sổ con x vượt quá một ngưỡng nào đó thì đó là khuôn mặt; ngược lại thì không phải là khuôn mặt.

## Thuật toán Adaboost (Adaptive boost)

### Sơ bộ về ý tưởng

Kỹ thuật Boosting: nguyên tắc cơ bản của Boosting là kết hợp các bộ phân lớp yếu (hay các bộ phân lớp cơ sở) để tạo nên một bộ phân lớp mạnh. Các bộ phân lớp yếu này thậm chí chỉ cần nhỉnh hơn phương pháp random một chút. Bằng cách này, ta nói bộ phân lớp đã được “boost.”

Adaboost là một thuật toán sử dụng kỹ thuật Boosting được đưa ra bởi Freund và Schapire vào 1996. Thuật toán này được sử dụng rất phổ biến cho bài toán phát hiện khuôn mặt.

Điểm cải tiến của Adaboost là ta sẽ **gán cho mỗi mẫu một trọng số**. Ý nghĩa của việc gán trọng số như sau:

Ở mỗi vòng lặp của quá trình huấn luyện, khi một bộ phân lớp yếu yi đã được xây dựng, ta sẽ tiến hành cập nhật trọng số cho các mẫu. Việc cập nhật này được tiến hành như sau: ta sẽ tăng trọng số của các mẫu bị phân lớp sai bởi bộ phân lớp yếu yi và giảm trọng số của các mẫu được phân lớp đúng bởi yi. Bằng cách này, ở vòng lặp kế, ta sẽ xây dựng bộ phân lớp yếu yi+1 theo hướng: **tập trung vào các mẫu bị phân lớp sai bởi bộ phân lớp yếu trước đó**.

Cuối cùng, để có được bộ phân lớp mạnh, ta sẽ kết hợp tuyến tính các bộ phân lớp yếu đã tìm được lại với nhau. Mỗi bộ phân lớp yếu sẽ được đánh một trọng số tương ứng với độ tốt của bộ phân lớp yếu đó.

Đến đây, có lẽ ta đã hiểu được ý nghĩa của từ “Adaptive” (thích nghi) trong tên gọi của thuật toán này: các bộ phân lớp yếu tạo thành một chuỗi, trong đó bộ phân lớp yếu sau sẽ tập trung giải quyết các mẫu bị phân lớp sai bởi các bộ phân lớp yếu trước đó.

### Thuật toán

Cho tập ảnh huấn luyện (x1, t1), …, (xn, tn) với ti {-1, 1}

1. Khởi tạo hệ số trọng cho mỗi mẫu huấn luyện: với n = 1, 2, …, N.
2. For m = 1, …, M:

(a) Xây dựng bộ phân lớp yếu ym:

+ Với mỗi đặc trưng j, xây dựng một bộ phân lớp yj với độ lỗi:

(1.0)

với nếu và = 0 nếu ngược lại.

+ Chọn bộ phân lớp yj với độ lỗi nhỏ nhất ta được ym.

(b) Cập nhật trọng số:

+ Tính:

(1.1)

và:

(1.2)

+ Cập nhật trọng số:

(1.3)

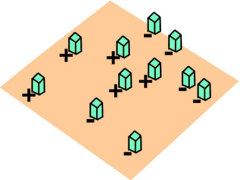
1. Bộ phân lớp mạnh cuối cùng:

(1.4)

### Hiểu thuật toán

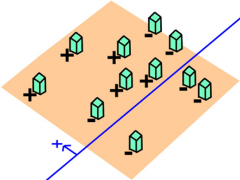
#### Mức sơ bộ

Thuật toán bắt đầu bằng việc khởi tạo trọng số cho các mẫu huấn luyện. Các trọng số này được khởi tạo bằng nhau. Các trọng số này cho thuật toán biết độ quan trọng của mẫu.



Ở mỗi vòng lặp, ta làm 2 việc:

* Thứ 1: tìm bộ phân lớp yếu ym dựa vào độ lỗi nhỏ nhất.

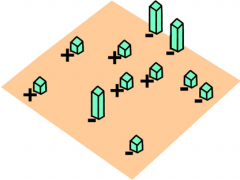
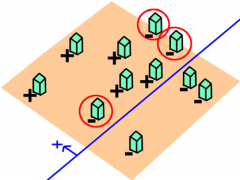


Công thức tính độ lỗi:

với nếu và = 0 nếu ngược lại.

Công thức này đơn giản chỉ là tính tổng các trọng số của các mẫu bị phân lớp sai.

* Thứ 2: cập nhật trọng số theo nguyên tắc: ta sẽ tăng trọng số cho các mẫu hiện đang bị phân lớp sai và giảm trọng số cho các mẫu hiện đang được phân lớp đúng. Bằng cách này, ở vòng lặp kế ta sẽ xây dựng bộ phân lớp yếu theo hướng tập trung giải quyết các mẫu bị phân lớp sai bởi bộ phân lớp yếu trước đó.



Trong đó:

với:

Ta nhận thấy nếu mẫu được phân lớp đúng thì trọng số không thay đổi; nếu mẫu bị phân lớp sai thì:

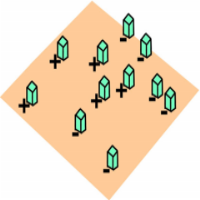
Thấy ngay

Thật vậy, giả sử , nghĩa là tỉ lệ phần sai lớn hơn phần đúng. Điều này mâu thuẫn với điều kiện bộ phân lớp yếu là tối ưu (có độ lỗi nhỏ nhất) vì rằng chỉ cần đổi chiều của mặt phẳng phân lớp lại là ta có ngay một bộ phân lớp yếu tốt hơn.

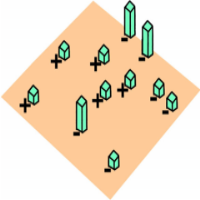
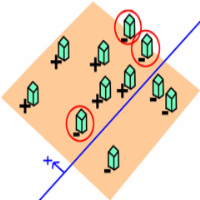
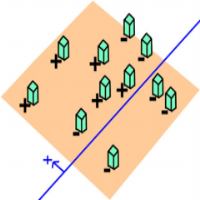
Và vì nên : trọng số của các mẫu bị phân lớp sai được tăng lên.

Để trực quan, ta hãy quan sát thuật toán thông qua chuỗi các hình vẽ dưới đây:

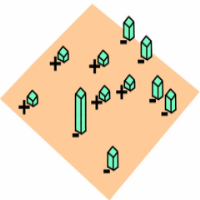
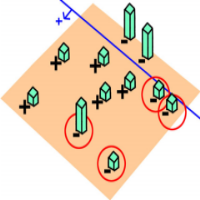
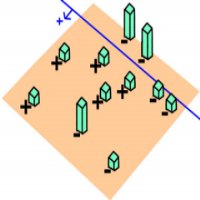
* Khởi tạo trọng số cho các mẫu:



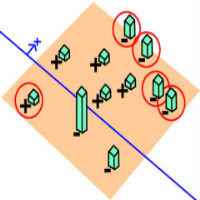
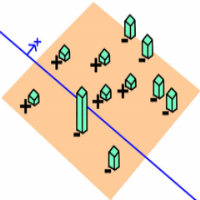
* Vòng lặp thứ 1:



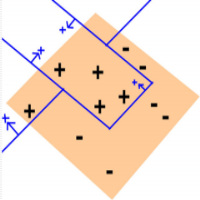
* Vòng lặp thứ 2:



* Vòng lặp thứ 3:



* Kết hợp các bộ phân lớp yếu lại:



Cuối cùng, kết hợp tuyến tính các bộ phân lớp yếu lại ta được một bộ phân lớp mạnh:

Nhận thấy mức độ đóng góp của mỗi bộ phân lớp yếu vào bộ phân lớp mạnh được quyết định bởi :

Dễ thấy tỉ lệ nghịch với mà thể hiện độ lỗi ứng với bộ phân lớp ym. Như vậy, độ lỗi càng thấp, tức bộ phân lớp càng tốt thì mức độ đóng góp của nó vào bộ phân lớp mạnh cuối cùng sẽ càng nhiều.

Nói thêm về điều kiện dừng: trong thực tế cài đặt, người ta sử dụng một giá trị ngưỡng là tỉ lệ nhận dạng sai tối đa (max false positive) để làm điều kiện dừng. Qua các vòng lặp, tỉ lệ nhận dạng sai của bộ phân lớp mạnh sẽ giảm dần. Đến một lúc nào đó, tỉ lệ này nhỏ hơn tỉ lệ nhận dạng sai tối đa và ta sẽ dừng thuật toán.

#### Mức chi tiết

Ở đây, ta sẽ chứng minh về toán học để thấy được vì sao lại có được các công thức trong thuật toán Adaboost.

Friedman (2000) đã đưa ra một cách hiểu rất đơn giản về thuật toán Adaboost. Ý tưởng sơ bộ là tối ưu hóa liên tiếp (sequential minimization) hàm lỗi mũ.

Ta có hàm lỗi mũ:

trong đó:

* là bộ phân lớp được tạo bởi sự kết hợp tuyến tính của các bộ phân lớp cơ sở:

với

Mục tiêu của ta là minimize E theo các biến: hệ số trọng và các tham số của bộ phân lớp cơ bản .

Ở đây, thay vì tối ưu hóa toàn cục, ta sẽ tiến hành tối ưu hóa dần dần (sequential minimization.)

Giả sử ta đã tìm được m-1 bộ phân lớp đầu. Như vậy, ở đây ta cần minimize E theo và .

Ta có:

với (1.5)

Đặt:

* Tm là tập các mẫu được phân lớp đúng bởi ym.
* Mm là tập các mẫu bị phân lớp sai bởi ym.

Có ngay:

Thêm bớt ta được:

(1) Minimize E theo : dẫn đến minimize:

Đây chính là công thức (1.0)

(2) Minimize E theo :

Cho đạo hàm = 0, ta có:

Đặt ta có:

Suy ra:

Suy ra tiếp:

Đến đây, ta có được công thức (1.2)

Bây giờ, ta sẽ chứng minh công thức cập nhật trong số (1.3)

Dựa vào (1.5) có ngay:

mà:

nên:

Vì với mọi mẫu, lượng là giống nhau nên ta có thể bỏ nó đi.

Vậy ta đã chứng minh xong công thức (1.3)

Và cuối cùng, dễ có công thức (1.4).

Như vậy, đến đây ta đã chứng minh xong thuật toán Adaboost.

# THỰC NGHIỆM

# KẾT LUẬN