Università degli studi di Cagliari
Dipartimento di Matematica e Informatica
Corso di Laurea Magistrale in Matematica

Il problema dell'Edge Detection

Studenti: Marco Ratto & Alessia Pistis

RICERCA OPERATIVA

Prof. M. Di Francesco
Prof. E. Gorgone

A.A. 2021/2022

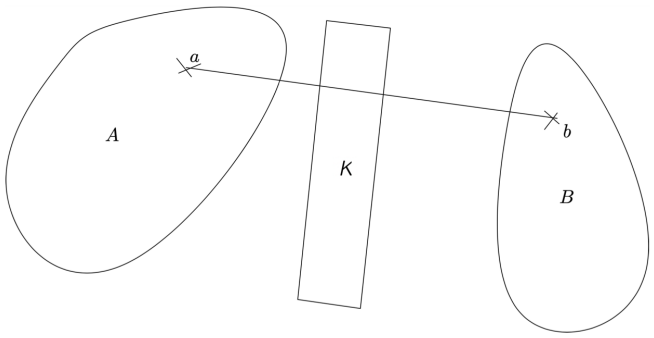
Introduzione

L'Edge Detection (Identificazione dei Bordi) fa parte di un insieme di metodi matematici che hanno come obiettivo quello di identificare i bordi e le curve di un'immagine nella quale c'è una variazione brusca dell'intensità luminosa.

La risoluzione del problema di determinare il contorno tra due regioni con differenti proprietà dei livelli di grigio si basa, dal punto di vista matematico, sulla classificazione di punti.
Il processo di classificazione consiste nella definizione di una funzione che, dato un insieme, assegna ad ogni suo elemento un'etichetta utilizzata per catalogarlo come appartenente a una certa classe.

Il primo passo per costruire un algoritmo di classificazione è quello di definire le caratteristiche delle classi nelle quali si vogliono separare gli elementi dell'insieme sotto studio.

Per prima cosa introduciamo il concetto di separabilità:
Un insieme convesso $\mathbf{K} \subset \mathbb{R}^n$ è un insieme separatore se dati due insiemi convessi $\mathbf{A}, \mathbf{B} \subset \mathbb{R}^n$, ogni segmento che congiunge un punto di $\mathbf{A}$ e un punto di $\mathbf{B}$ interseca $\mathbf{K}$, cioè se:
$\mathbf{K} \cap [a, b] = \varnothing$, $\forall a \in \mathbf{A}, \forall b \in \mathbf{B}$.



Nel problema dell'Edge Detection viene utilizzata in particolare la classificazione supervisionata. Si parla di classificazione supervisionata quando si parte da un insieme di punti
$P = \{(x, y) | x \in \mathbb{R}^n, y \in \{-1, 1\}\}$ dove i punti con etichetta -1 appartengono alla classe $\mathbf{A}$ e quelli con etichetta +1 a $\mathbf{B}$.

Vogliamo utilizzare i dati in input per definire un insieme separatore $\mathbf{K}$ per le due classi $\mathbf{A}$ e $\mathbf{B}$ e poi usarlo per definire una funzione di classificazione $f_K : \mathbb{R}^n \longrightarrow \{-1, 1\}$ che permette di classificare qualunque punto di $\mathbb{R}^n$.
Dato un insieme $X \subset \mathbb{R}^n \times \{-1, 1\}$ vogliamo determinare l'iperpiano che separa le due classi di punti $\mathbf{A}$ e $\mathbf{B}$. Definisco un iperpiano H come: $H = \{(\omega, \gamma) | x^\intercal \omega + \gamma,\ \omega \in \mathbb{R}^n, \gamma \in \mathbb{R}^n\}$.

Se $\mathbf{A}$ e $\mathbf{B}$ sono linearmente separabili allora esiste almeno un iperpiano H i cui punti soddisfano le condizioni:

1. $a_i^\intercal \omega + \gamma \leq -1$
2. $b_j^\intercal \omega + \gamma \geq 1$

$\forall\, a_i \in \mathbf{A}, b_j \in \mathbf{B}$.

Tra tutti gli iperpiani si sceglie quello con il margine maggiore, cioè quello che separata maggiormente i punti.

Formuliamo il problema come:

$$\begin{cases} min_{\omega,\gamma} \quad \|\omega\|^2 \\ t.c. \quad y_i(x_i^\intercal \omega + \gamma) \geq 1 \\ y_i \in \{-1, 1\} \end{cases} \tag{1}$$

Quando le due classi **A** e **B** non sono linearmente separabili allora si introducono delle variabili slack che fanno in modo che il problema sia ancora ammissibile.

$$\begin{cases} min_{\omega,\gamma} \quad \|\omega\|^2 + c_1 \Sigma_i \xi_i^- + c_2 \Sigma_j \xi_j^+ \\ a_i \omega + \gamma - \xi_i \leq -1 \\ b_j \omega + \gamma + \xi_i^+ \geq 1 \\ \xi^\pm \geq 0 \end{cases}$$

I parametri c_1 e c_2 vengono modificati per ottenere un miglioramento del risultato.
Se le due classi **A** e **B** non sono linearmente separaribili in $\mathbb{R}^n$ allora si parla di classificazione non lineare. In questo caso lo spazio originale di dimensione $\mathbb{R}^n$ viene mappato in uno spazio con una dimensione maggiore $\mathbb{R}^k$, con $k > n$.
Il problema viene riformulato utilizzando una funzione kernel $\varphi : \mathbb{R}^n \longrightarrow \mathbb{R}^k$ nel modo seguente.

$$\begin{cases} min_{\omega,\gamma} \quad \|\omega\|^2 + c_1 \Sigma_i \xi_i^- + c_2 \Sigma_j \xi_j^+ \\ t.c. \quad y_i(\varphi(x_i^\intercal)\omega + \gamma) \geq 1 \\ \quad y_i \in \{-1, 1\} \\ \quad \omega \in \mathbb{R}^n \\ \quad \gamma \in \mathbb{R} \end{cases}$$

In particolare sfrutteremo la classificazione ellissoidale e utilizzeremo il modello matematico di SVM con il kernel quadratico:

$$\begin{cases} min_{\omega,\gamma} \quad \|\omega\|^2 + c_1 \Sigma_i \xi_i^- + c_2 \Sigma_j \xi_j^+ \\ \varphi(a_i)\omega + \gamma - \xi_i \leq -1 \\ \varphi(b_j)\omega + \gamma + \xi_i^+ \geq 1 \\ \xi^\pm \geq 0 \end{cases}$$

Con $\varphi(x) = (x_1^2, ..., x_n^2, \sqrt{2}x_1x_2, ..., \sqrt{2}x_1^n, ..., \sqrt{2}x_nx_{n-1}, \sqrt{2c}x_1, ..., \sqrt{2c}x_n, c) \in \mathbb{R}^n$,
$N = \frac{1}{2}(n+1)(n+2)$.

Implementazione

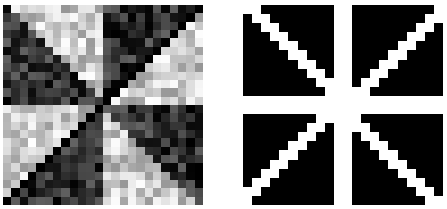
Per svolgere questo lavoro abbiamo utilizzato il programma MATLAB e il pacchetto Yalmip, utilizzando le funzioni svm e svmk.
Nel primo caso abbiamo implementato in questo modo:

```
function [w,b,ea,eb] = svm(A,B,c1,c2)
% A partire da due insiemi di punti, determina l'iperpiano separatore
% risolvendo un problema di programmazione lineare.
% In output restituisce i coefficienti w e b che descrivono l'iperpiano e
% gli errori relativi alle due classi di punti

% I parametri c1  e c2, se non specificati, vengono messi di default uguali
% a 1.
if nargin < 4
    c2 = 1;
end

if nargin < 3
    c1 = 1;
end

na = size(A,1);           % numero dei punti di classe A
nb = size(B,1);           % numero dei punti di classe B

k = size(A,2);            % dimensione dei punti


w = sdpvar(k,1);            % variabili
b = sdpvar(1,1);

ea = sdpvar(na,1);          % variabili
eb = sdpvar(nb,1);

% Insieme dei vincoli:
Constraints = [A*w + b - ea <= -1, B*w + b + eb >= 1, ea >= 0, eb >= 0];

% Funzione obiettivo:
Objective = norm(w)^2 + c1*norm(ea,1) + c2*norm(eb,1);

% Utilizzo il pacchettp Yalmip per calcolare la soluzione ottima
optimize(Constraints, Objective)

% Converto le variabili in scalari/vettori
w = value(w);
b = value(b);
ea = value(ea);
eb = value(eb);
```

Analogamente nel caso della funzione svmk abbiamo:

```
function [w,b,ea,eb] = svmk(A,B,c1,c2)
% Analogo a SVM, viene applicata alle trasformazioni phi(A) e phi(B)
if nargin < 4
    c2 = 1;
end

if nargin < 3
    c1 = 1;
end

phiA = phi(A);
phiB = phi(B);

na = size(A,1);           % numero dei punti di classe A
nb = size(B,1);           % numero dei punti di classe B

k = size(A,2);            % dimensione dei punti di partenza
K = (k+1)*(k+2)/2;        % dimensione dei punti trasformati

w = sdpvar(K,1);
b = sdpvar(1,1);

ea = sdpvar(na,1);


eb = sdpvar(nb,1);

Constraints = [phiA*w + b - ea <= -1, phiB*w+b+eb >= 1,ea >= 0,eb >= 0

Objective = norm(w)^2 + c1*norm(ea,1) + c2*norm(eb,1);

optimize(Constraints, Objective)

w = value(w);
b = value(b);
ea = value(ea);
eb = value(eb);
```

Per addestrare il programma siamo partiti da un'immagine 24x24 della quale conosciamo le etichette. Abbiamo utilizzato in particolare queste due immagini:



```
% Edge detection

Im = imread('trainingim.png');
Lab = imread('traininglab.png');

mi = size(Im,1);
ni = size(Im,2);

% Creo il vettore Y contenente le etichette di ogni  pixel:
% se il pixel è bianco (255), è di bordo (etichetta 1)
% se è nero (0), non è di bordo (etichetta -1)
Y = [];
for i = 1:size(Lab)
    for j = 1:size(Lab)
        if Lab(i,j) == 255
            Y = [Y;1];
        end
        if Lab(i,j) == 0
            Y = [Y;-1];
        end
    end
end
```

Alla matrice dell'immagine associamo una matrice X che ha in ogni riga i valori assoluti delle differenze di luminosità tra il pixel della riga e quelli che si trovano attorno.

```
function X = ImVec(A)
% Questa funzione associa ad un'immagine (matrice) un'altra matrice, che ha
% in ogni riga i valori assoluti delle differenze di luminosità tra il
% pixel a cui è associata la riga e i suoi pixel adiacenti

m = size(A,1);
n = size(A,2);

% Inizializzo la matrice
X = zeros((m-2)*(n-2),4);

% Con due cicli for riempio le entrate di X con i valori desiderati
for j = 2:(n-1)
    for i = 2:(m-1)
        X((m-2)*(j-2)+(i-1),:) = [abs(A(i,j)-A(i-1,j)), abs(A(i,j)-A(i,j+1)),...
            abs(A(i,j)-A(i+1,j)), abs(A(i,j)-A(i,j-1))];
    end
end
```

Dato che sono note le etichette, dividiamo i vettore della matrice X in questo modo:

1. i vettori associati a punti di bordo (matrice B)
2. i vettori associati a punti non di bordo (matrice A)

Una volta ottenuti i due gruppi di punti A e B, applichiamo su questi la SVM sia con il kernel che senza kernel, in modo tale da trovare i coefficienti $\omega\, e\, \gamma$ dell'iperpiano separatore rappresentato da $x^\intercal\omega + \gamma = 0$.
Creiamo quindi una funzione che prenda la matrice X dei vettori delle differenze delle luminosità e che possa predire le etichette di una qualsiasi immagine (della quale non sono note), utilizzando l'iperpiano determinato in precedenza.
Avremo che:

1. Se $x^\intercal\omega + \gamma < 0$, x non è di bordo.
2. Se $x^\intercal\omega + \gamma > 0$, x è di bordo.

```
function Ypred = predict(X,w,b)
% Calcola per ogni vettore della matrice X da che parte si trova rispetto
% all'iperpiano della SVM.
n = size(X,1);
Ypred = [];

for i = 1:n
    if dot(w,X(i,:)) + b < 0
        Ypred(i) = 1;
    end
    if dot(w,X(i,:)) + b > 0
        Ypred(i) = -1;
    end
end
Ypred = Ypred';
```

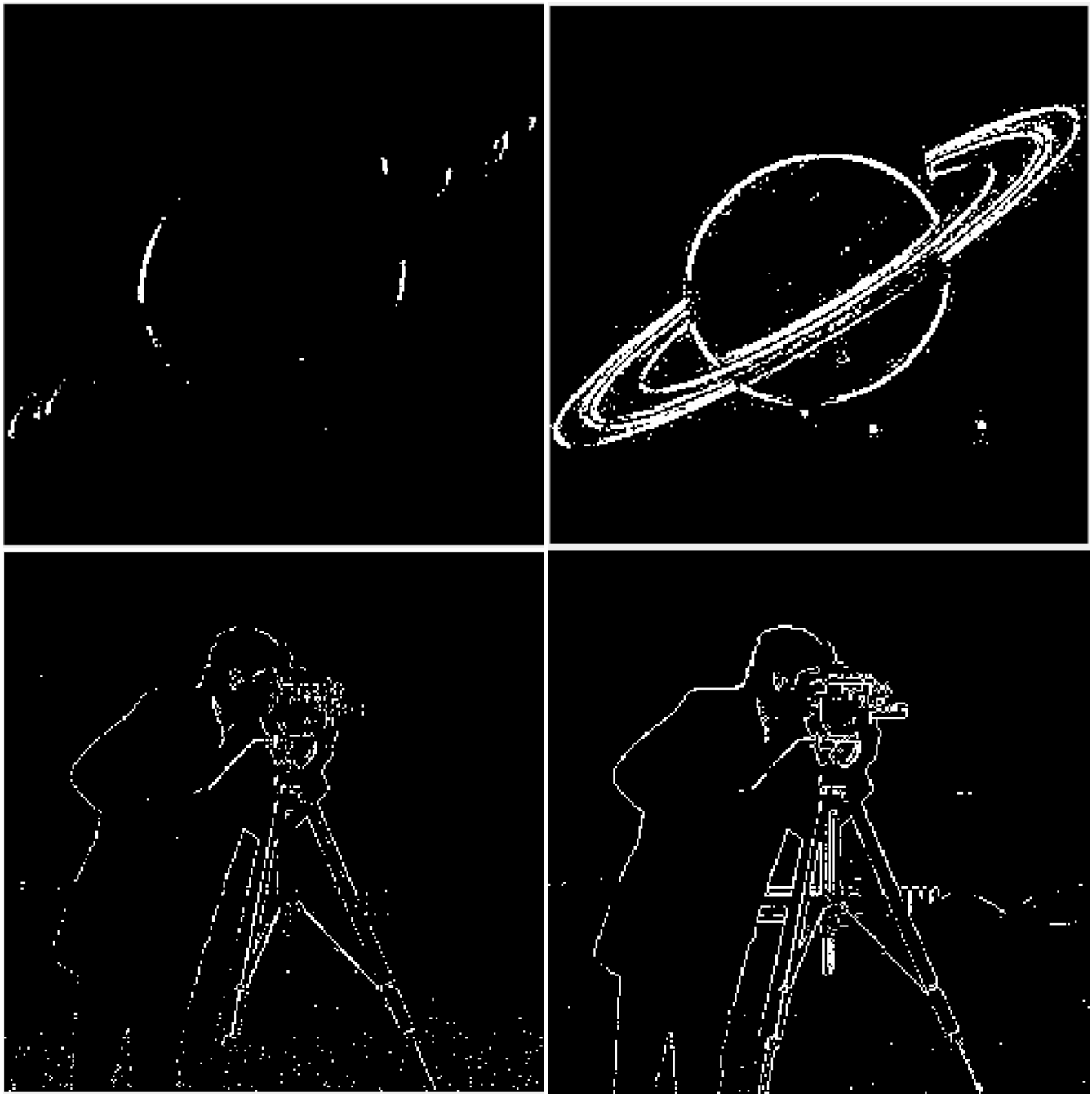
A questo punto testiamo la precisione del nostro predittore sull'immagine test, in modo da poter confrontare le etichette predette con quelle che già conoscevamo. Successivamente, utilizzando lo stesso iperpiano, proviamo a determinare i bordi di nuove immagini.

Risultati

Per l'applicazione abbiamo utilizzato diverse immagini e confrontato i risultati, partendo dall'utilizzo di immagini più semplici.
Per prima cosa abbiamo considerato:

Utilizzando le immagini iniziali poste sopra, tramite il codice sottoriportato abbiamo ottenuto i seguenti risultati:

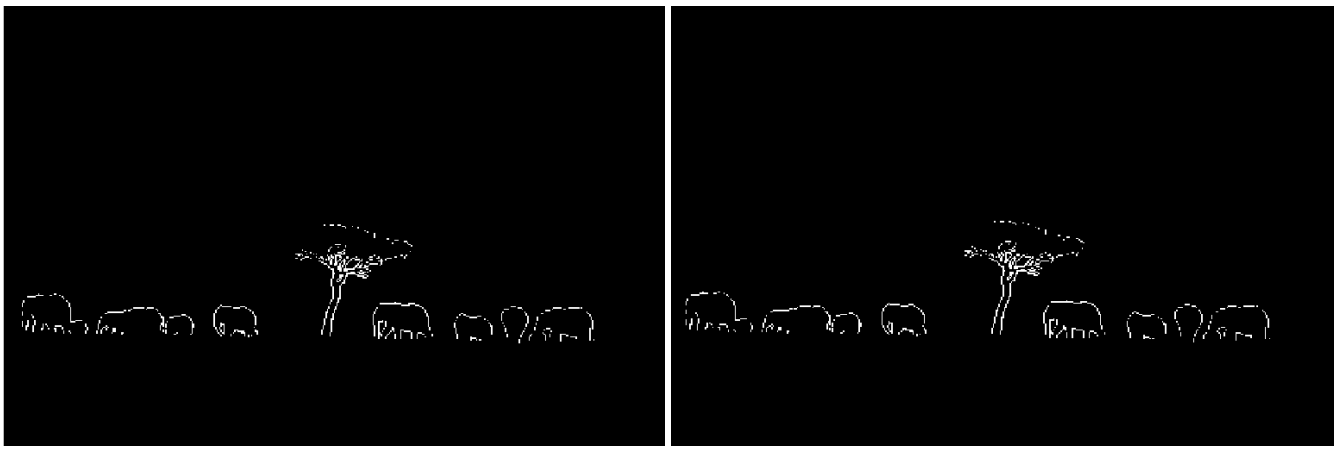
```
% Immagine pianeta
planet2 = imread('planet.jpeg');        % leggo l'immagine
planet = rgb2gray(planet2);             % la trasformo in scala di grigi (era a colori)

Xplanet = ImVec(planet);                % calcolo la 'matrice X'
Xplanet = double(Xplanet);
Yplanet = predict(Xplanet,w,b);         % faccio la predizione usando i coefficienti
Yplanetk = predict(phi(Xplanet),wk,bk); % della SVM con e senza kernel

% Rimetto in forma di matrice e converto in immagine
Implan = mat2gray(reshape(Yplanet,size(planet,1)-2,size(planet,2)-2));
Implank = mat2gray(reshape(Yplanetk,size(planet,1)-2,size(planet,2)-2));
imshow(Implank)

% Ripeto gli stessi passaggi per le altre immagini
```



Nelle immagini poste a destra è stata utilizzata la funzione kernel mentre in quelle a sinistra no.

A questo punto abbiamo lavorato su immagini più complesse, ad esempio:

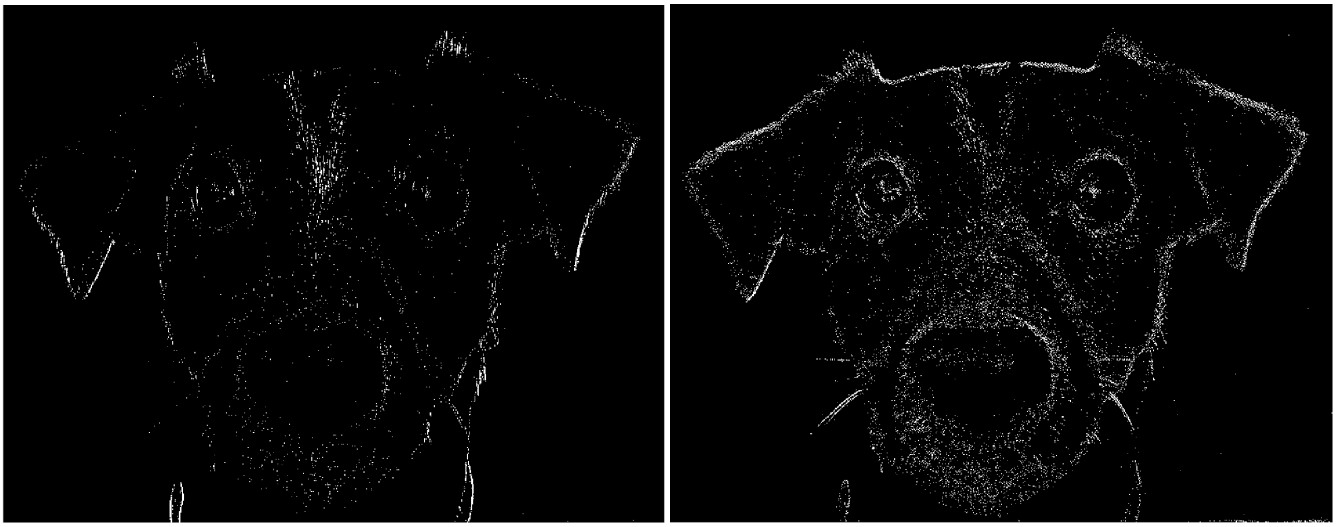
Come nei casi precedenti abbiamo eseguito due prove. L'immagine a destra è stata ottenuta con l'utilizzo della funzione kernel.



Analogamente abbiamo trattato la seguente immagine:

```
%Immagine cane
dog = imread('dog.gif');

Xdog = ImVec(dog);

Xdog = double(Xdog);
Ydog = predict(Xdog,w,b);
Ydogk = predict(phi(Xdog),wk,bk);
Imdog = mat2gray(reshape(Ydog,size(dog,1)-2,size(dog,2)-2));
Imdogk = mat2gray(reshape(Ydogk,size(dog,1)-2,size(dog,2)-2));
imshow(Imdogk)
```

Ottenendo i risultati:



Si può notare in tutti i casi in analisi che si ottiene un notevole miglioramento con l'utilizzo della funzione kernel.