Sergiu Nedevschi
Florin Oniga
Ion Giosan
Robert Varga
Raluca Brehar
Andra Petrovai

SISTEME DE RECUNOAȘTERE A FORMELOR

Îndrumător de laborator
Ediția I-a

UTPRESS
Cluj-Napoca, 2023
ISBN 978-606-737-638-8

Sergiu NEDEVSCHI
Florin ONIGA
Ion GIOSAN

Robert VARGA
Raluca BREHAR
Andra PETROVAI

SISTEME DE RECUNOAȘTERE A FORMELOR

Îndrumător de laborator

Ediția 1

UTPRESS
Cluj-Napoca, 2023
ISBN 978-606-737-638-8

Editura UTPRESS
Str. Observatorului nr. 34
400775 Cluj-Napoca
Tel.: 0264-401.999
e-mail: utpress@biblio.utcluj.ro
www.utcluj.ro/editura

Director: ing. Dan COLȚEA

Recenzia: Prof.dr.ing. Dorian Gorgan
Prof.dr.ing. Vasile Dădârlat

Pregătire format electronic on-line: Gabriela Groza

ISBN 978-606-737-638-8

Cuprins

Prefață

Introducere

Îndrumătorul de laborator reprezintă materialul de bază pentru pregătirea lucrărilor practice la disciplina Sisteme de Recunoaștere a Formelor și este destinat studenților de anul IV, ciclul de licență, domeniul Calculatoare și Tehnologia Informației din cadrul Facultății de Automatică și Calculatoare. În prezenta ediție sunt studiate o serie de metode din domeniile de Procesări de Imagini, Recunoaștere a Formelor și Învățare Automată. Principalul scop este de a dezvolta sisteme de percepție și reprezentare automată a scenei din imagini.

Suport electronic

Materiale suplimentare se găsesc la următorul link:

http://cv.utcluj.ro/index.php/teaching.html

Printre materiale se numără: proiectul de start (pentru diferite versiuni de Visual Studio); o introducere în librăria OpenCV; materiale suplimentare care sunt necesare pentru implementarea sarcinilor practice.

Software

Software-ul recomandat pentru finalizarea sarcinilor practice este Visual Studio 2013 sau o versiune ulterioară. Fișierele soluției Visual Studio sunt furnizate pentru mai multe versiuni. De asemenea, este posibil să configurați manual proiectul și să legați biblioteca OpenCV. Pentru aceasta, consultați documentația bibliotecii. Orice alt editor sau IDE cu suport C++ poate fi folosit, cum ar fi Visual Studio Code, Eclipse sau CLion. Problemele pot fi programate în orice limbaj cu suport OpenCV, cum ar fi Python sau Java, dar în acest îndrumător se utilizează C++.

Precondiții

Următoarele cunoștințe sunt necesare pentru a înțelege și a finaliza cu succes materialul prezentat:

- Algebră liniară – operații cu matrici, sisteme liniare, descompunere cu valori proprii;
- Geometrie analitică și trigonometrie – ecuații parametrice pentru curbe/suprafețe;
- Analiză – funcții multivariate, derivate parțiale, minime locale și globale;
- Statistică și teoria probabilității – caracterizarea variabilelor aleatoare;
- Structuri de date și algoritmi – liste de puncte/vectori, sortare obiecte arbitrare;
- Programare C++ – fișiere text de intrare/ieșire, funcții și argumente, acces la memorie și alocare dinamică

1 Metoda celor mai mici pătrate

1.1 Obiective

Scopul acestei lucrări de laborator este de a introduce metoda celor mai mici pătrate, prin care se doreşte potrivirea unei linii la o mulţime de puncte 2D. Se prezintă atât o soluţie iterativă cât şi o formulă directă pentru mai multe modele.

1.2 Fundamente teoretice

Se dă o mulţime de puncte bidimensionale de forma (x_i, y_i) unde $i = \{1,2, \dots, n\}$. Obiectivul este găsirea ecuaţiei liniei care se potriveşte cel mai bine la aceste puncte. Pentru aceasta se va utiliza regresia liniară (metoda celor mai mici pătrate).

1.2.1 Model 1 – Model liniar cu pantă și termen liber

La orice metodă de potrivire primul pas constă în definirea modelului. La început vom folosi un model liniar care conţine un termen pentru pantă şi un termen liber. Exprimăm componenta y în funcţie de x folosind funcţia:

$$f(x) = \theta_0 + \theta_1 x$$

De obicei acest model este folosit pentru rezolvarea problemei. Însă această reprezentare nu poate să modeleze linii verticale. Totuşi vom porni de la acest model simplu. Se poate forma un vector care va conţine toţi parametrii $\boldsymbol{\theta} = [\theta_0, \theta_1]^{\mathrm{T}}$ (termenul liber şi coeficientul lui x).

Metoda curentă adoptă o funcţie de cost care însumează erorile pătrate dintre estimator şi valorile originale. Modelul ideal va fi obţinut când funcţia de cost atinge minimul global:

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{n}(f(x_i) - y_i)^2$$

De ce pătratic? Putem motiva această alegere prin presupunerea că eroarea în date urmează o distribuţie gaussiană. O observaţie importantă este că această funcţie penalizează erorile doar în direcţia y

şi nu foloseşte distanţele punctelor la dreaptă. Pentru a minimiza funcţia de cost se calculează derivatele parţiale:

$$\frac{\partial}{\partial\theta_0}J(\theta) = \sum_{i=1}^{n}(f(x_i) - y_i)$$

$$\frac{\partial}{\partial\theta_1}J(\theta) = \sum_{i=1}^{n}(f(x_i) - y_i)\,x_i$$

Funcţia de cost atinge minimul global când derivatele parţiale sunt nule.

Metoda gradient descent

O metodă generală pentru a găsi soluţia este *gradient descent*. Deoarece gradientul arată direcţia în care funcţia creşte cel mai mult, făcând un pas în direcţia opusă valoarea funcţiei descreşte. Dacă facem mai multe iteraţii şi controlăm mărimea paşilor vom atinge minimul global. Deoarece funcţia de cost este pătrătică există un singur minimim global care va fi găsit de această abordare.

Inițial se aleg valori aleatorii dar diferite de 0 pentru parametri. Apoi se calculează gradientul în punctul curent:

$$\nabla J(\boldsymbol{\theta}) = \left[\frac{\partial J(\boldsymbol{\theta})}{\partial\theta_0}, \frac{\partial J(\boldsymbol{\theta})}{\partial\theta_1}\right]^T$$

Şi apoi se aplică următoarea regulă până la convergenţă:

$$\boldsymbol{\theta}_{new} = \boldsymbol{\theta} - \alpha\nabla J(\boldsymbol{\theta}),$$

unde α este rata de învăţare şi este aleasă astfel încât funcţia de cost să descrească la fiecare iterație. Algoritmul se oprește când se ajunge la un număr maxim de iterații sau valoarea erorii este mai mică decât un prag. Algoritmul *gradient descent* se poate sumariza astfel:

Algoritmul Gradient Descent

Se aleg valori inițiale pentru θ_0 și θ_1
Se alege o valoare inițială pentru rata de învățare α
Se alege numărul maxim de iterații *max_iter*
Se alege un prag T pentru eroare
for iter = 1: *max_iter*

$$\frac{\partial}{\partial\theta_0}J(\theta) = \sum_{i=1}^{n}(f(x_i) - y_i)$$

$$\frac{\partial}{\partial\theta_1}J(\theta) = \sum_{i=1}^{n}(f(x_i) - y_i)\,x_i$$

$$\theta_0 := \theta_0 - \alpha\frac{\partial}{\partial\theta_0}J(\theta)$$

$$\theta_1 := \theta_1 - \alpha\frac{\partial}{\partial\theta_1}J(\theta)$$

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{n}(f(x_i) - y_i)^2$$

if $J(\theta) < \mathrm{T}$
  break

Metoda directă

Metoda gradient descent este potrivită atunci când este dificil să găsim în mod analitic rădăcinile sistemului de ecuaţii. Pentru modelul curent se poate determina foarte uşor soluţia finală. Ecuaţiile pentru derivatele parţiale egale cu 0 se aduc la următoarea formă:

$$\begin{cases} \theta_0 n + \theta_1\sum_{i=1}^{n}x_i = \sum_{i=1}^{n}y_i \\ \theta_0\sum_{i=1}^{n}x_i + \theta_1\sum_{i=1}^{n}x_i^2 = \sum_{i=1}^{n}x_i y_i \end{cases}$$

care este un sistem de ecuaţii liniare în două necunoscute şi poate fi rezolvat. Se obţin valorile:

$$\begin{cases} \theta_1 = \dfrac{n\sum_{i=1}^{n}x_i y_i - \sum_{i=1}^{n}x_i\sum_{i=1}^{n}y_i}{n\sum_{i=1}^{n}x_i^2 - (\sum_{i=1}^{n}x_i)^2} \\ \theta_0 = \dfrac{1}{n}\left(\sum_{i=1}^{n}y_i - \theta_1\sum_{i=1}^{n}x_i\right) \end{cases}$$

Ecuaţiile normale - O soluţie generală sub formă vectorială

În majoritatea cazurilor sistemul de ecuaţii se poate formula sub formă vectorială. Fie matricea A formată din liniile $[1\ x_i]$ şi fie vectorul coloana $\boldsymbol{b}$ care conţine toate valorile y_i. Folosind aceste notaţii se doreşte minimizarea normei:

$$\|A\boldsymbol{\theta} - \boldsymbol{b}\|^2 = (A\boldsymbol{\theta} - \boldsymbol{b})^T(A\boldsymbol{\theta} - \boldsymbol{b})$$

Această formulă se generalizează foarte uşor pentru mai multe dimensiuni. În acest caz soluţia poate fi obţinută prin formula:

$$\boldsymbol{\theta}_{opt} = (A^T A)^{-1} A^T \boldsymbol{b}$$

Pentru mai multe detalii şi demonstraţie consultaţi [1].

1.2.2 Model 2 – Forma normală a dreptei

Adoptăm un alt model pentru a rezolva problemele amintite în partea anterioară. Acest model este capabil să trateze toate cazurile cu succes. Considerăm o parametrizare a unei linii de forma:

$$xcos(\beta) + ysin(\beta) = \rho$$

Aceasta descrie o linie cu normală unitară de formă $[cos(\beta), sin(\beta)]$ care se află la o distanţă ρ față de origine.

Scriem acum funcţia de cost care va fi suma distanţelor la pătrat de la fiecare punct la dreaptă:

$$J(\beta, \rho) = \frac{1}{2}\sum_{i=1}^{n}(x_i cos(\beta) + y_i sin(\beta) - \rho)^2$$

Observăm că aceasta reprezintă eroarea adevărată pe care trebuie să o minimizăm şi că funcţia de cost pentru modelul 1 măsoară doar discrepanţa pe axa y.

Derivatele parţiale au următoarea formă:

$$\frac{\partial J}{\partial \beta} = \sum_{i=1}^{n}(x_i cos(\beta) + y_i sin(\beta) - \rho)\big(-x_i sin(\beta) + y_i cos(\beta)\big)$$

$$\frac{\partial J}{\partial \rho} = -\sum_{i=1}^{n}(x_i cos(\beta) + y_i sin(\beta) - \rho)$$

Şi în acest caz putem să obţinem o soluţie directă, însă este mult mai dificil să rezolvăm sistemul de ecuații. Formulele pentru cei doi parametri sunt:

$$\beta = -\frac{1}{2} atan2\left(2 - \frac{2}{n}\sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i \,,\, \sum_{i=1}^{n}(y_i^2 - x_i^2) + \frac{1}{n}\left(\sum_{i=1}^{n} x_i\right)^2 - \frac{1}{n}\left(\sum_{i=1}^{n} y_i\right)^2\right)$$

$$\rho = \frac{1}{n}\left(\cos(\beta)\sum_{i=1}^{n} x_i + sin(\beta)\sum_{i=1}^{n} y_i\right)$$

1.2.3 Model 3 – Forma standard a dreptei

Există încă o posibilitate care oferă o rezolvare generală. Dacă folosim o parametrizare cu 3 parametri:

$$ax + by + c = 0$$

Această parametrizare tratează corect liniile verticale fiindcă acestea se modelează considerând *b=0*. Funcţia de cost se defineşte ca:

$$J(a,b,c) = \frac{1}{2}\sum_{i=1}^{n}(ax_i + by_i + c)^2$$

care poate fi scris vectorial sub forma unei norme care trebuie minimizat:

$$J(a,b,c) = (A\boldsymbol{\theta})^T A\boldsymbol{\theta}$$

unde *A* este o matrice cu *nx3* elemente, fiecare linie conţine $(x_i, y_i, 1)$ şi $\boldsymbol{\theta} = [a,b,c]^T$ este vectorul de parametri (un vector coloană). Observăm că funcţia de cost diferă faţă de cea definită la partea cu *ecuaţia normală*.

Utilizarea unui model cu 3 parametri are două consecinţe relevante: se poate modela orice linie, dar pentru o linie avem mai multe parametrizări echivalente. Pentru a rezolva problema ambiguităţii impunem restricţia ca $\boldsymbol{\theta}$ să aibă normă unitară. Soluţia la această problemă utilizează descompunerea cu valori singulare (*Singular Value Decomposition – SVD*). Descompunem A în trei matrici:

$$A = USV$$

unde *U* şi *V* sunt ortogonale şi *S* conţine valori nenule doar pe diagonală (valori singulare). Soluţia se citeşte ca şi ultima coloana din matricea *V* care corespunde la valoarea singulară cea mai mică. Pentru mai multe detalii consultaţi [2].

$$\theta_{opt} = V(:,n)$$

1.3 Consideraţii practice

Descărcați proiectul Visual Studio. Acesta conține câteva exemple de funcții și include biblioteca OpenCV pentru procesarea imaginilor. Se pot utiliza următoarele fragmente de cod pentru rezolvarea exercițiilor:

Citirea din fişier:

```
FILE* f = fopen("filename.txt","r");
float x,y;
fscanf(f,"%f%f", &x,&y);
fclose(f);
```

Crearea unei imagini color:

```
Mat img(height, width, CV_8UC3); //8 bit unsigned 3 channel
```

Accesarea pixelului de pe rândul *i* şi coloana *j*:

```
Vec3b pixel = img.at<Vec3b>(i,j); //byte vector with
//3 elements
```

Modificarea pixelului de la poziţia (*i*, *j*):

```
img.at<Vec3b>(i,j)[0] = 255; //blue
img.at<Vec3b>(i,j)[1] = 255; //green
img.at<Vec3b>(i,j)[2] = 255; //red
```

Desenarea unei linii care trece prin 2 puncte. Primul punct se află pe rândul *y1* si coloana *x1*, iar al doilea pe rândul *y2* și coloana *x2*.

```
line(img, Point(x1, y1), Point(x2, y2), Scalar(B,G,R));
```

Afişarea imaginii:

```
imshow("title", img);
waitKey();
```

1.4 Activitate practică

1. Citiţi datele de intrare din fişierele ataşate. Prima linie conţine numărul de puncte. Liniile următoare conţin perechi (x,y).
2. Afişaţi punctele pe o imagine albă de dimensiune 500x500. Pentru vizibilitate mai bună se poate trasa un cerc, sau un pătrat în jurul fiecărui punct. Aveţi în vedere convenţia pentru sistemul de coordonate al imaginii. Unele puncte pot avea coordonate negative. Acestea ori nu se afişează ori se translatează graficul. Metoda în sine nu este afectată de faptul că punctele au coordonate negative.
3. Folosiţi *modelul 1* şi formulele pentru a calcula parametrii. Vizualizaţi linia şi comparaţi rezultatul cu soluţia iterativă de la pasul anterior.
4. Folosiţi *modelul 2* şi metoda directă pentru a calcula parametrii. Vizualizaţi linia şi comparaţi rezultatul cu soluţia iterativă de la pasul anterior.
5. Opţional, utilizaţi modelul 1 cu gradient descent. Vizualizaţi poziţia liniei la fiecare pas şi tipăriţi valorile funcţiei de cost.

Trebuie să alegeţi o rată de învăţare care asigura descreşterea funcţiei de cost.

6. Opţional, utilizaţi modelul 2 şi gradient descent pentru a găsi parametrii. Vizualizaţi poziţia liniei la fiecare pas şi tipăriţi valorile funcţiei de cost. Trebuie să alegeţi o rată de învăţare care asigură descreşterea funcţiei de cost.
7. Opţional, utilizaţi modelul 3 pentru a găsi ecuaţia dreptei.

1.5 Exemple de rezultate

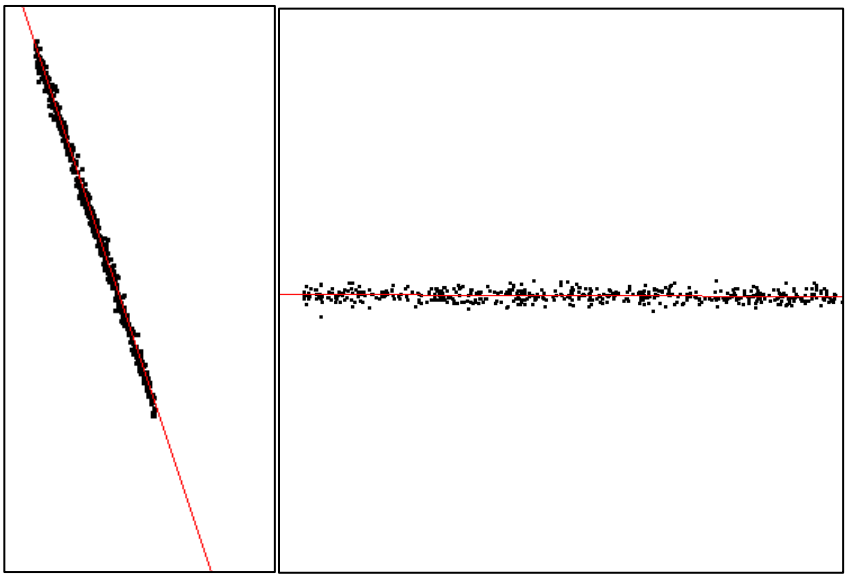


Figura 1.1 – Rezultate obținute folosind modelul 2 pe datele din fișierele *points1.txt* și *points2.txt*

1.6 Referinţe

[1] Stanford Machine Learning CS229 - lecture notes 1 – https://see.stanford.edu/materials/aimlcs229/cs229-notes1.pdf

[2] Tomas Svoboda - Least-squares solution of Homogeneous Equation - http://cmp.felk.cvut.cz/cmp/courses/XE33PVR/WS20072008/Lectures/Supporting/constrained_lsq.pdf

2 RANSAC

2.1 Obiective

Scopul acestei lucrări de laborator este de a introduce o metodă robustă de potrivire a unui model la un set de puncte (care poate conține și zgomote), denumită RANSAC.

2.2 Fundamente teoretice

Random Sample Consensus (RANSAC) - consensul eșantioanelor aleatorii - este o paradigmă pentru potrivirea unui model la date experimentale, introdusă de Martin A. Fischler şi Robert C. Bolles în 1981 [1].

După cum au declarat Fischler şi Bolles "Procedura RANSAC este opusul tehnicilor convenționale de filtrare: în loc să folosim cât mai multe date posibile pentru a obține o soluție inițială, şi apoi să eliminăm punctele invalide, RANSAC foloseşte un set iniţial de date cât mai mic posibil şi apoi măreşte acest set cu date valide atunci când este posibil."

Algoritmul RANSAC este descris în cele ce urmează [2]:

Obiectiv: Potrivirea robustă a unui model la un set de date *S* care conține puncte valide (fără zgomot) și invalide (cu zgomot).

Algoritm RANSAC

1. Selecția aleatorie a unui eșantion de puncte *s* din mulțimea *S* și instanţierea modelului din acest eșantion.

2. Determinarea mulţimii de date S_i care conţine puncte situate la o distanţă mai mică decât un prag *t* de model. Mulţimea S_i este mulţimea de consens a eşantionului, şi definește punctele din *S* care satisfac modelul.

3. Dacă dimensiunea lui S_i (numărul de puncte care satisfac modelul) este mai mare decât un prag T, algoritmul se termină. Opţional, se poate estima din nou modelul folosind toate punctele din S_i.

4. Dacă dimensiunea lui S_i este mai mică decât T, se selectează un nou subset şi se repetă paşii anteriori.

5. După N încercări, se selectează mulţimea de consens S_i cu cele mai multe puncte, şi modelul este reestimat folosind toate punctele din această mulţime.

Definiția parametrilor este următoarea:

- $\boldsymbol{s}$ – dimensiunea subsetului selectat pentru potrivirea modelului, adică numărul de puncte;
- $\boldsymbol{S}$ – setul de puncte;
- $\boldsymbol{S_i}$ – submulțimea punctelor valide (fără zgomot) pentru pasul i, sau setul de suport;
- $\boldsymbol{t}$ – valoarea prag pentru distanța maximă admisă față de model;
- $\boldsymbol{T}$ – valoarea prag pentru semnalizarea unei potriviri a datelor suficient de bună;
- $\boldsymbol{N}$ – numărul maxim de iterații;

2.2.1 RANSAC pentru linii

Considerați următoarea problemă: fiind dat un set S de puncte 2D, găsiți dreapta care minimizează suma distanțelor punctelor față de linie (regresie ortogonală). Datele pot fi contaminate (puncte zgomotoase sau incorecte), astfel potrivirea unei linii folosind **Metoda celor mai mici pătrate** pe toate punctele setului de date S ar duce la rezultate incorecte – vezi figura 2.1-a.

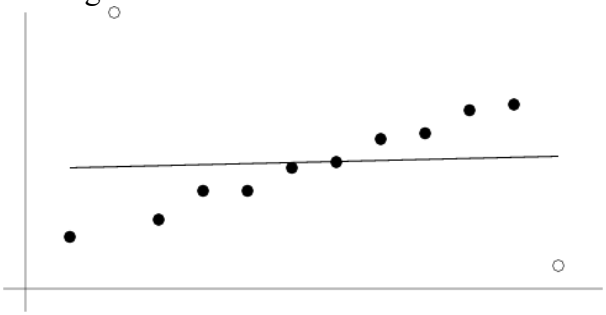


Figura 2.1-a – Linie obținută prin Metoda celor mai mici pătrate aplicată întregului set de date

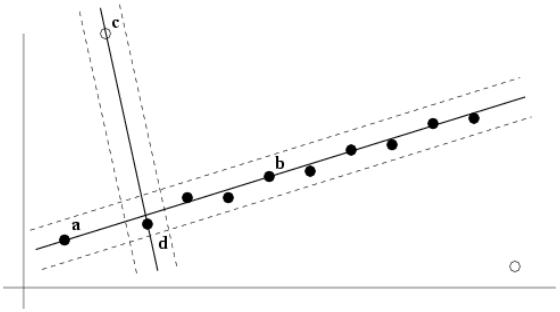


Figura 2.1-b – Doua linii posibile luate in considerare de RANSAC

Algoritmul RANSAC poate fi aplicat pentru potrivirea unei drepte la un set de puncte. Problema este următoarea: fiind dată o mulțime de puncte 2D, găsiți linia care minimizează suma distanțelor perpendiculare (regresie ortogonală), în condițiile în care nici unul din punctele valide nu deviază de la linie cu mai mult de *t* unități. Există de fapt două probleme: potrivirea unei linii la date, și clasificarea datelor în puncte valide și puncte de zgomot. Pragul *t* este selectat în funcție de zgomotul propriu al măsurătorii.

Primul pas este selectarea a două puncte în mod aleator, iar aceste puncte vor defini o linie. Mulțimea suport (consens) pentru această linie este dată de punctele care sunt la mai puțin de o distanță prag de această linie. Selecția aleatorie este repetată de mai multe ori, şi linia care are cele mai multe puncte suport este considerată potrivirea cea mai robustă. Intuitiv, dacă unul din cele două puncte alese pentru potrivirea liniei este parte a zgomotului, linia estimată nu va avea o mulțime suport foarte mare.

Dacă măsurăm calitatea unei linii prin mărimea mulțimii suport, avem avantajul de a favoriza cele mai bune linii de la început. Astfel, linia (*a*, *b*) din Figura 2.1-b are o mărime a suportului de 10, pe când linia (*c*, *d*) are un suport de doar 2. Putem deduce că *c* sau *d* este un punct zgomot.

În continuare, abordăm câteva întrebări privind selectarea parametrilor:

- **De ce metoda este randomizată?** Încercarea exhaustivă a tuturor subseturilor este posibilă numai pentru un set de date mic. De exemplu, pentru un set de date de dimensiunea $|S| = n$ avem $n(n-1)/2$ perechi de puncte pentru a verifica posibilele linii. Problema devine rapid greu de rezolvat pentru

valorile lui n de ordinul 10^5. Dacă modelul este potrivit folosind mai mult de 2 puncte, subseturile posibile sunt și mai mari. Selectând în mod repetat două puncte aleatorii evităm să verificăm toate submulțimile posibile.

- **Câte încercări ar trebui să facem?** Valoarea pentru numărul de iterații N ar trebui aleasă astfel încât să existe o probabilitate p suficient de mare ca cel puțin în una dintre cele N încercări toate punctele selectate să fie valide. Pentru a calcula N se consideră următoarele:
    - q – este probabilitatea estimată ca un punct să fie valid
    - q^s – este probabilitatea ca toate cele s puncte să fie valide
    - $1 - q^s$ – este probabilitatea ca cel puțin un punct să fie invalid
    - $(1 - q^s)^N$- este probabilitatea ca în fiecare dintre cele N încercări să existe cel puțin un punct invalid
    - $p = 1 - (1 - q^s)^N$ – este probabilitatea ca cel puțin o încercare să fie fără puncte invalide
    - Valoarea lui N poate fi calculată pe baza unei valori fixe pentru un p selectat
      $N = \log(1 - p) / \log(1 - q^s)$

- **Cum se alege pragul distanței t?** Am dori să alegem pragul distanței t, astfel încât un punct să fie valid cu o probabilitate dată q. Pentru aceasta avem nevoie de distribuția de probabilitate pentru distanța unui punct valid față de model (modelul erorii de măsurare). În practică, pragul distanței este de obicei ales empiric. Totuși, dacă se presupune că eroarea de măsurare este gaussiană cu medie zero și deviație standard σ, atunci valoarea pentru t poate fi setată la 3σ.

- **Cât de mare este un consens acceptabil?** O regulă generală este să oprim algoritmul dacă dimensiunea setului de consens este similară cu numărul de puncte valide care se presupune că sunt în setul de date. Având în vedere proporția de valori valide q, pentru n puncte de date, dimensiunea consensului trebuie sa fie mai mare ca $T = q \cdot n$.

2.2.2 Modelul dreptei

Ecuaţia unei drepte care trece prin două puncte distincte (x_1, y_1) şi (x_2, y_2) este dată de:

$$(y_1 - y_2)x + (x_2 - x_1)y + x_1y_2 - x_2y_1 = 0$$

Distanţa de la un punct (x_0, y_0) la o dreaptă definită de $ax + by + c = 0$ este:

$$d = \frac{|ax_0 + by_0 + c|}{\sqrt{a^2 + b^2}}$$

2.3 Consideraţii practice

Deschiderea unei imagini cu niveluri de gri:

```
Mat img = imread("filename", IMREAD_GRAYSCALE);
```

Crearea unei imagini cu niveluri de gri:

```
     Mat dst(height, width, CV_8UC1);
     // 8 bit unsigned char 1 channel
```

Accesarea pixelului de la rândul *i* coloana *j*:

```
uchar pixel = img.at<uchar>(i,j); //unsigned
char type
```

Un punct negru la poziţia `(i,j)` din imagine corespunde la un punct *p* cu coordonatele `x=j, y=i`:

```
if (img.at<uchar>(i,j)==0){
     Point p; p.x = j; p.y = i;
}
```

Modificarea pixelului de pe rândul *i* coloana *j*:

```
img.at<uchar>(i,j) = 255; //white
```

Selectarea unui punct aleator dintr-un vector de *n* puncte (este necesară includerea fişierului *stdlib.h*):

```
Point p = points[rand()%n];
```

Desenarea unei linii care trece prin două puncte (*x1, y1*) şi (*x2, y2*):

```
line(img, Point(x1, y1), Point(x2, y2),
Scalar(B,G,R));
```

Afisarea imaginii:

```
imshow("title", img);
waitKey();
```

2.4 Activitate practică

1. Construiţi lista de puncte prin găsirea tuturor punctelor negre din imaginea de intrare.
2. Calculaţi parametrii necesari *N* şi *T* pornind de la valorile *t=10, p=0.99, q=0.8, s=2*. Pentru *points1.bmp* folosiţi *q=0.3*.
3. Aplicaţi algoritmul RANSAC:
    a. Alegeţi două puncte diferite;
    b. Determinaţi ecuatia dreptei care trece prin cele două puncte alese;
    c. Calculaţi distanţa tuturor punctelor la dreaptă;
    d. Număraţi punctele valide (*inliers*) ale mulțimii de consens;
    e. Memoraţi parametrii liniei (*a, b, c*) dacă linia curentă are cele mai multe puncte valide până acum;
    f. Stabiliţi condiţiile de oprire (mulţime de consens suficient de mare sau număr maxim de iteraţii).
4. Desenaţi linia optimală găsită de metodă.

2.5 Referințe

[1] Robert C. Bolles, Martin A. Fischler: *A RANSAC-Based Approach to Model Fitting and Its Application to Finding Cylinders în Range Data*, 1981

[2] Richard Hartley, Andrew Zisserman: *Multiple View Geometry în Computer Vision,* 2003

3 Detecția dreptelor cu transformata Hough

3.1 Obiective

Obiectivul principal al acestei lucrări este studierea transformatei Hough pentru detecția dreptelor în imagini binare.

3.2 Fundamente teoretice

Transformata Hough este o metodă care rezolvă o problemă clasică din viziunea artificială: găsirea dreptelor într-o imagine binară ce conţine o mulţime de puncte de interes (de exemplu puncte de muchii). Metoda directă de a calcula drepte din fiecare pereche de puncte are o complexitate computaţională ridicată de $O(n^2)$, şi nu este aplicabilă pentru un număr mare de puncte. Transformata Hough a fost propusă şi patentată de Peter Hough [1], şi în varianta iniţială, a fost o metodă de timp real pentru a număra câte puncte sunt plasate pe fiecare dreaptă posibilă dintr-o imagine. Această metodă se bazează pe reprezentarea dreptei sub formă pantă-termen liber ($y=ax+b$), şi pe construirea unui spaţiu parametric, numit şi acumulator Hough. Pentru fiecare punct de interes din imagine se calculează toate dreptele posibile care trec prin el, şi se incrementează elementele din spaţiul parametric. Dreptele relevante sunt localizate în maximele locale ale spaţiului parametric.

Această reprezentare este sub-optimală, deoarece nu este mărginită. Pentru a reprezenta toate dreptele posibilele din imagine, panta şi termenul liber trebuie să varieze în domeniul $-\infty$ şi $+\infty$. Modificările propuse de Duda şi Hart [2] au făcut transformata Hough populară în domeniul viziunii artificiale. Principala problemă legată de parametri nemărginiţi a fost rezolvată prin parametrizarea normală a dreptei în sistemul de coordonate polar. Parametrizarea normală a unei drepte constă în reprezentarea dreptei prin vectorul normal (perpendicular) şi distanţa de la origine. Reprezentarea normală se mai numeşte şi reprezentarea ρ-θ (Figura 3.1).

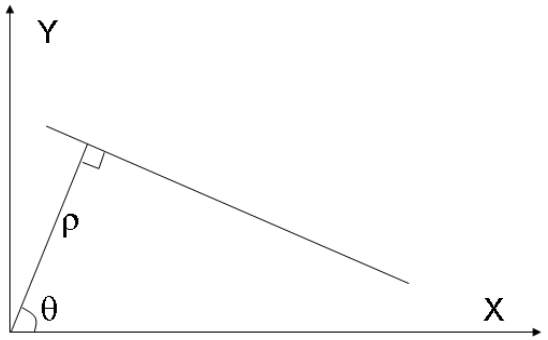


Figura 3.1. Reprezentarea dreptei în sistemul de coordonate polar. Dreapta se află la o distanţă ρ faţă de origine şi normala face unghiul θ cu axa Ox.

O dreaptă poate fi reprezentată folosind parametrii ρ și θ:

$$\rho = x\cos(\theta) + y\sin(\theta) \qquad (1)$$

Parametrii ρ și θ pot avea o infinitate de valori. Pentru a descreşte complexitatea computaţională se introduce cuantizarea parametrilor ρ și θ, proces prin care se constrâng parametrii să aibă un număr finit de valori. Parametrii ρ si θ au un interval de variaţie limitat deoarece imaginea are o dimensiune finită. Valoarea maximă pentru ρ este diagonala imaginii. În funcție de intervalul ales pentru θ, există două configuraţii echivalente pentru domeniul parametrilor. Prima este cea propusă în articolul original iar noi o vom adopta pe a doua.

$$\begin{aligned} &1.\ \theta \in [-90^{\circ}, 90^{\circ})\ or\ \theta \in [0, 180^{\circ}),\ \ \rho \in [-\rho_{\max}, +\rho_{\max}] \\ &2.\ \theta \in [0, 360^{\circ}),\ \ \rho \in [0, +\rho_{\max}] \end{aligned} \qquad (2)$$

De asemenea, pentru fiecare dintre cei doi parametri ai dreptei se stabileşte un nivel de cuantizare care depinde de acurateţea cerută (de exemplu: pasul pentru ρ poate fi de 10, 1, 0.5 pixeli etc., iar pasul pentru θ poate fi de 10 grade, 1 grad, 0.5 grade etc.).

În general printr-un punct din planul imagine (x_0, y_0), putem defini un set de drepte care trec prin punctul respectiv la diferite unghiuri θ. Având (x_0, y_0) și θ, parametrul dreptei ρ se poate calcula conform ecuatiei (2). Pentru a detecta dreptele dintr-o imagine, algoritmul Hough folosește o schemă de votare: fiecare punct de interes (x_0, y_0), de exemplu pixelii de pe muchii, votează pentru toate dreptele cu parametrii (ρ, θ) care trec prin punctul respectiv. Dreptele care

primesc cele mai multe voturi sunt considerate dreptele finale detectate de algoritmul Hough. Un exemplu este prezentat in Figura 3.2.

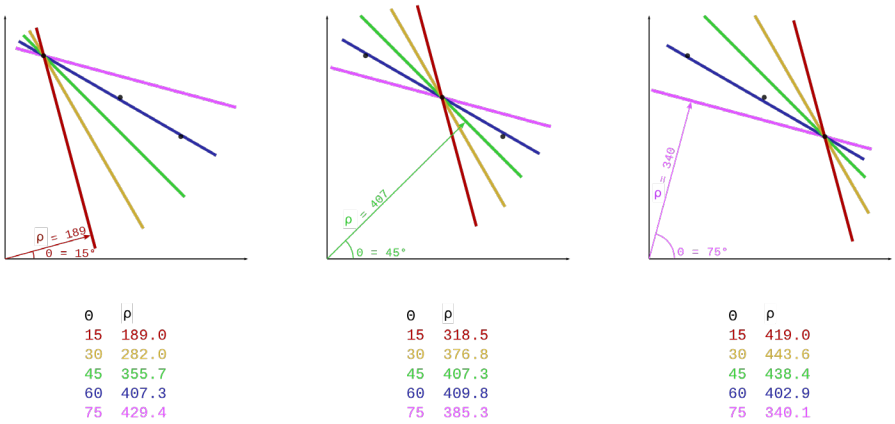


Figura 3.2 Se consideră 3 puncte de interes marcate cu negru. Se dorește determinarea dreptelor din imagine. Pentru fiecare punct avem un set de drepte având parametrii (ρ, θ) care trec prin acel punct. Fiecare punct votează pentru dreptele care trec prin acel punct. Observăm că în toate cazurile, pentru dreapta cu θ=60°, valorile pentru ρ sunt similare. Se poate concluziona că punctele se află în vecinătatea dreptei albastre cu θ=60°, care va fi detectată de algoritmul Hough. Exemplu preluat din [4].

Pentru a stoca voturile, algoritmul Hough folosește un vector 2D (matrice), denumit acumulatorul Hough (H). Acesta reprezintă spaţiul parametrilor (ρ, θ) cuantizaţi ai dreptei. Paşii de cuantizare pentru ρ si θ sunt $\Delta\rho$ şi $\Delta\theta$, respectiv. Valorile lor maxime sunt ρ_{max} si θ_{max}. Atunci acumulatorul va avea o dimensiune de ($\rho_{max}/\Delta\rho$ x $\theta_{max}/\Delta\theta$). Operaţia de incrementare a unei locaţii Hough poate fi ponderată – de exemplu cu modulul gradientului. După ce am construit acumulatorul, dreptele relevante se extrag ca maxime locale ale acestuia. Un maxim local este un punct unde valoarea din acumulator este mai mare decât toate valorile dintr-o vecinătate (pătratică). Algoritmul Hough este următorul:

Algoritmul Hough

1. Se iniţializează fiecare celulă din H cu 0
2. Se creează acumulatorul H:
   Pentru fiecare punct de muchie P(x, y) din imagine
      Pentru fiecare θ de 0 la θ_{max} (cu un pas de Δθ)
         Se calculează $\rho = x\cos(\theta) + y\sin(\theta)$
         Dacă $\rho \in [0, +\rho_{max}]$ se incrementează H(ρ, θ)

3. Se găsesc primele k maxime locale din H:
   Se verifică dacă $H(\rho, \theta)$ este maxim local într-o fereastră n x n centrată pe celula respectivă:
   Pentru fiecare θ de la n la $\theta_{max} - n$ (cu un pas de $\Delta\theta$)
   Pentru fiecare ρ de la n la $\rho_{max} - n$ (cu un pas de $\Delta\rho$)
   Se stochează tripletul $[\rho, \theta, H(\rho, \theta)]$ dacă $H(\rho, \theta)$ are cea mai mare valoare din fereastra n x n și $H(\rho, \theta) > T$
4. Se ordonează descrescător tripletele stocate în funcție de $H(\rho,\theta)$ și se returnează primele k drepte (ρ, θ)

Parametrii algoritmului Hough sunt următorii:

- P - o imagine binară
- θ_{max} - valoarea maximă pe care o poate lua θ
- ρ_{max} - valoarea maximă pe care o poate lua ρ
- $\Delta\theta$ - pasul pentru θ
- $\Delta\rho$ - pasul pentru ρ
- n – dimensiunea ferestrei pentru determinarea maximului local
- T – numărul minim de voturi (sau echivalent numărul minim de puncte) pentru ca o dreaptă să fie considerată validă
- k – numărul de drepte dorit

Un exemplu de detecție a liniilor pe baza transformatei Hough se prezintă în Figura 3.3. Domeniul de variație al parametrilor pentru acest exemplu este [0, 360) grade pentru θ, și [0, 144] pixeli pentru ρ. Pasul parametrilor este de 1 grad pentru θ și de 1 pixel pentru ρ.

Alegerea unui nivel de cuantizare adecvat este foarte importantă. Dacă se face o cuantizare prea fină, rezoluţia creşte odată cu timpul de procesare, şi cresc şi şansele ca puncte aparent colineare să incrementeze celule diferite din acumulator (acest lucru va cauza detecţii multiple ale aceleiaşi drepte, sau fragmentarea unei drepte în mai multe părţi).

Deşi transformata Hough se foloseşte cel mai des pentru detecţia dreptelor, ea poate fi folosită şi pentru detecţia curbelor mai complexe, atât timp cât o parametrizare adecvată este disponibilă. Duda şi Hart [2] au propus detecţia cercurilor, folosind un spaţiu de parametri tridimensional şi transformând fiecare punct într-un con circular în spaţiul parametric (toate cercurile posibile ce conţin respectivul punct).

Mai târziu, Ballard a generalizat transformata Hough pentru a detecta orice formă non-analitică [3].

3.3 Detalii de implementare

Folosiți cea mai simplă configurație posibilă pentru cuantizarea parametrilor: 1 pixel pentru ρ şi 1 grad pentru θ. Folosiți a doua variantă pentru domeniul de variație al parametrilor conform ecuațiilor (3). Astfel vom detecta dreptele care au unghiul $\theta \in [0, 360°)$ cu un pas $\Delta\theta$=1 și $\rho \in [0, \rho_{max}]$ cu un pas $\Delta\rho$=1. Dimensiunea acumulatorului Hough va fi de $\rho_{max}+1$ rânduri și $\theta_{max}=360$ de coloane, unde ρ_{max} este diagonala imaginii:

$$\rho_{max} = \sqrt{height^2 + width^2}$$

Se declară acumulatorul Hough ca o matrice cu elemente de tip întreg:

```
Mat Hough(ρmax+1,360,CV_32SC1);
```

Acumulatorul se inițializează cu 0 folosind:

```
Hough.setTo(0);
```

Pentru a calcula ρ conform ecuației (2) este necesară transformarea θ din grade în radiani:

θ_{rad} = θ * CV_PI/180

Apoi se poate calcula ρ:

ρ = x cos(θ_{rad}) + y sin(θ_{rad})

Acumulatorul se modifică (numai dacă $\rho \in [0, +\rho_{max}]$) folosind:

```
Hough.at<int>(ρ, θ)++;
```

Pentru a afișa acumulatorul sub forma unei imagini cu nivele de gri, acesta trebuie normalizat ca valorile să fie între 0-255. Se poate folosi funcția `normalize` din biblioteca OpenCV sau se poate normaliza

pe baza valorilor maxime din acumulator folosind secțiunea de cod următoare:

```
Mat houghImg;
Hough.convertTo(houghImg,                CV_8UC1,
255.f/maxHough);
```

unde `maxHough` reprezintă valoarea maximă din acumulator după construire. Calculele ulterioare se fac pe acumulatorul original.

Pentru a localiza maximele locale din acumulator, se va testa pentru fiecare element dacă este maxim local într-o fereastră pătratică ($n \times n$) centrată pe element. Un element este maxim local dacă este mai mare decât toate elementele din fereastră. Rețineți acele elemente care sunt maxime locale şi care au valoarea mai mare decât un prag. Ordonaţi elementele reținute, şi păstrați primele k vârfuri, care reprezintă cele mai mari maxime locale şi corespund la dreptele importante.

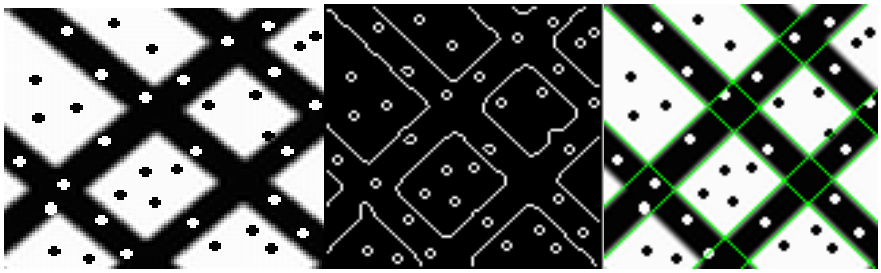
Următoarea structură vă permite stocarea maximelor locale şi sortarea lor cu un apel la metoda `sort` din biblioteca `algorithm`. Operatorul < a fost redefinit pentru a realiza sortarea descrescătoare a maximelor locale în funcţie de valorile din acumulatoare (`hval`).

```
struct peak{
     int theta, ro, hval;
     bool operator < (const peak& o) const {
          return hval > o.hval;
     }};
```

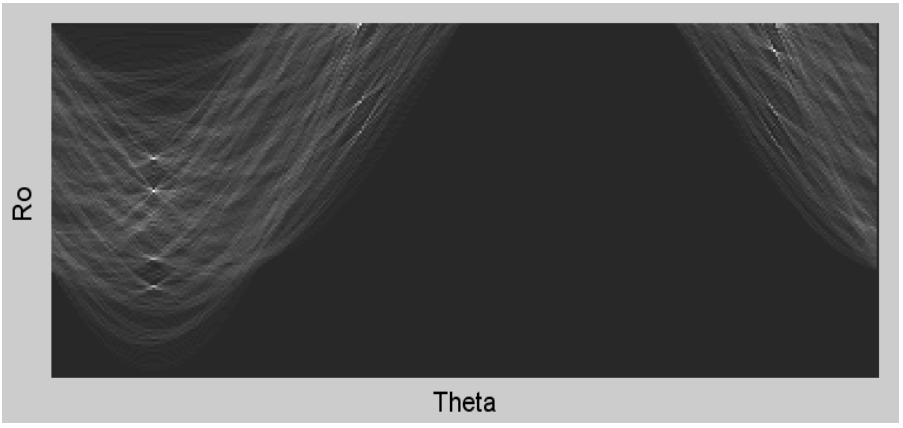
3.4 Activitate practică

1. Calculați acumulatorul Hough folosind imaginea de muchii. Afișați rezultatul sub forma unei imagini cu niveluri de gri.
2. Găsiți primele k maxime locale. Folosiți mărimi diferite pentru fereastra de suport, de exemplu de 3x3, 7x7 sau 11x11 (parametru pentru metodă) și diferite valori pentru pragul T.
3. Desenați liniile asociate cu aceste k vârfuri atât pe imaginea originală cât şi pe imaginea de muchii.

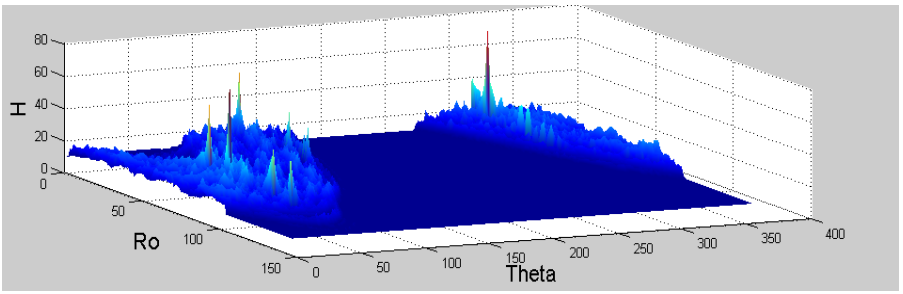
3.5 Exemple de rezultate



a. b. c.



d.



e.

Figura 3.3. **a**. Imagine cu un model cu muchii drepte, afectată de zgomot sare și piper **b**. Muchiile detectate cu un detector de muchii Canny, **c**. Cele mai relevante drepte sunt marcate cu verde, si ele se vor asocia cu cele mai relevante 8 maxime din acumulatorul Hough, **d.** Acumulatorul Hough afișat sub forma unei imagini cu niveluri de gri, **e.** Acumulatorul Hough afișat în 3D, folosind codificarea în culoare.

3.6 Referințe

[1] P. Hough, “Method and means for recognizing complex patterns”, US patent 3,069,654, 1962.
[2] R. O. Duda and P. E. Hart, "Use of the Hough Transformation to Detect Lines and Curves in Pictures," *Comm. ACM*, Vol. 15, pp. 11–15, 1972.
[3] D. H. Ballard, "Generalizing the Hough Transform to Detect Arbitrary Shapes", *Pattern Recognition*, Vol.13, No.2, p.111-122, 1981.
[4] https://en.wikipedia.org/wiki/Hough_transform

4 Transformata distanță (TD). Potrivirea modelelor folosind TD

4.1 Obiective

În acest laborator vom studia un algoritm care calculează transformata distanță a unei imagini binare (obiect și fundal). Această transformare permite evaluarea unui cost de potrivire a modelului unui obiect cunoscut (de exemplu un contur de pieton) și un obiect necunoscut, pentru a decide dacă obiectul necunoscut este sau nu similar cu obiectul model. Cu cât costul de potrivire este mai mic, cu atât obiectul necunoscut este mai asemănător modelului.

4.2 Fundamente teoretice

4.2.1 Transformata distanță

Transformata distanță, cunoscută și ca hartă de distanțe sau câmp de distanțe, este o reprezentare a unei imagini digitale. Harta asociază fiecărui pixel din imagine informația privitoare la distanța sa față de cel mai apropiat pixel obiect (sau obstacol). În practică, cel mai des întâlnit tip de pixel obiect este un punct de muchie.

Transformata distanță este un operator care se aplică în mod normal doar imaginilor binare. Rezultatul transformării este o imagine cu niveluri de gri, care seamănă cu imaginea intrare, dar în care intensitățile punctelor arată distanța față de cel mai apropiat punct obiect.

În imaginea următoare avem un exemplu de aplicare a transformatei distanță pe o imagine ce conține o formă dreptunghiulară. În imaginea din stânga, pixelii cu valoarea “0” sunt pixeli obiect (puncte de muchie), iar cei cu valoarea “1” puncte de fundal. În imaginea din dreapta, se vede rezultatul aplicării TD folosind metrica “tabla de șah”, unde fiecare valoare codifică distanța față de un punct de muchie.

0	0	0	0	0	0	0
0	1	1	1	1	1	0
0	1	1	1	1	1	0
0	1	1	1	1	1	0
0	1	1	1	1	1	0
0	1	1	1	1	1	0
0	0	0	0	0	0	0
Imaginea de intrare binară						

0	0	0	0	0	0	0
0	1	1	1	1	1	0
0	1	2	2	2	1	0
0	1	2	3	2	1	0
0	1	2	2	2	1	0
0	1	1	1	1	1	0
0	0	0	0	0	0	0
Transformata distanță						

Figura 4.1 – stânga: imaginea binară de intrare; dreapta: transformata distanță a imaginii; fiecare poziție conține distanța "tablă de șah" către cel mai apropiat punct de contur (valorile 0). [1]

De obicei transformata/harta este denumită pe baza metricii alese. De exemplu, se poate vorbi de Transformata Distanță Manhattan, dacă metrica folosită este distanța Manhattan.

Alte metrici sunt:

- Distanța Euclidiană:

1.41	1.0	1.41
1.0	0.0	1.0
1.41	1.0	1.41

- Distanța *City block* sau *Manhattan*:

2	1	2
1	0	1
2	1	2

- Distanța „tablă de șah"

1	1	1
1	0	1
1	1	1

Există mai mulți algoritmi pentru implementarea TD:

- Chamfer TD;
- Euclidian TD;
- Voronoi diagram TD.

Cea mai directă metodă de calculare a transformatei distanță este metoda *brute force*. Aceasta presupune calcularea pentru fiecare pixel de fundal distanțele către fiecare pixel de obiect. Apoi se ia minimul dintre aceste distanțe pentru fiecare pixel de fundal. Metoda este foarte ineficientă, prin urma vom studia o soluție mai eficientă, transformata distanță Chamfer, care aproximează distanța Euclideană, este o metodă simplă și foarte rapidă, necesitând doar două parcurgeri ale imaginii binare.

Algoritmul Transformata Distanță Chamfer

1. Se alege o mască de ponderi 3x3 care are valori proporționale cu distanțele Euclidiene față de elementul din mijloc. Cele mai simple și mai mici astfel de valori sunt 2 pentru deplasarea laterală și 3 pentru deplasarea diagonală. În acest fel distanțele obținute vor fi egale cu aproximativ dublul distanțelor euclidiene reale.

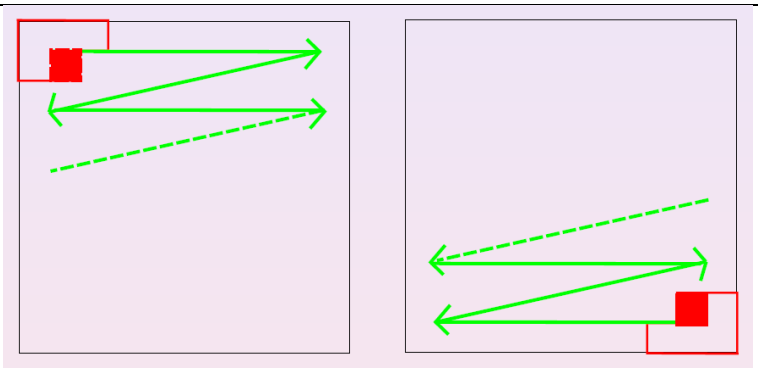
$$ponderi = \begin{bmatrix} 3 & 2 & 3 \\ 2 & 0 & 2 \\ 3 & 2 & 3 \end{bmatrix}$$

2. Transformata distanță are aceeași dimensiune ca imaginea de intrare și se inițializează cu valori de zero și valori infinit:

$$DT(i,j) = \begin{cases} 0, & (i,j) \in obiect \\ \infty, & (i,j) \notin obiect \end{cases}$$

   Dacă aplicăm TD pe o imagine binară, unde valoarea 0 înseamnă pixeli obiect și valoarea 255 codifică fundalul și dorim să obținem o imagine TD în tonuri de gri (8 biți/pixel), valoarea ∞ din algoritm ar trebui înlocuită cu valoarea 255.

3. O dublă parcurgere (prima dată sus-jos, stânga-dreapta, a doua oară jos-sus, dreapta-stânga) a imaginii (cu jumătățile corespunzătoare ale măștii, vezi figura de mai jos) este necesară pentru a actualiza distanța minimală. La prima traversare elementul central este comparat cu elementele galbene corespunzătoare vecine, iar la a doua traversare, cu elemente verzi:



În timpul parcurgerii imaginii sursă (direct și invers), se face următoarea actualizare pe imaginea TD:

$$TD(i,j) = \min_{(k,l)\in Mask}(TD(i+k,j+l) + ponderi(k,l))$$

4.2.2 Potrivirea de modele folosind TD

Dorim să calculăm un cost de similitudine între un model de obiect cunoscut și un obiect necunoscut. Considerăm că ambele obiecte au aceeași dimensiune.

Pașii pentru calcularea costului sunt:

1. Calcularea imaginii TD pentru modelul de obiect cunoscut;
2. [Opțional] Suprapunerea obiectului necunoscut peste imaginea TD calculată anterior prin translatarea obiectului necunoscut astfel încât centrul său de masă să fie același cu centrul de masă al obiectului model. Pentru simplitate, centrele de masă se estimează din punctele de pe contur. Centrul de masă (C_x, C_y) al obiectului descris de conturul **Ω** se poate aproxima după ecuația:

$$(C_x, C_y) = \left(\frac{1}{N}\sum_{p\,\in\,\Omega} p_x\,, \frac{1}{N}\sum_{p\,\in\,\Omega} p_y\right),$$

unde (p_x, p_y) reprezintă coordonatele x și y ale punctului de contur p, iar N este lungimea conturului.

3. Costul de similitudine (potrivire) este media tuturor valorilor pixelilor din imaginea TD care se află sub punctele de pe conturul obiectului necunoscut. Un cost mic de similitudine arată că obiectele sunt similare.

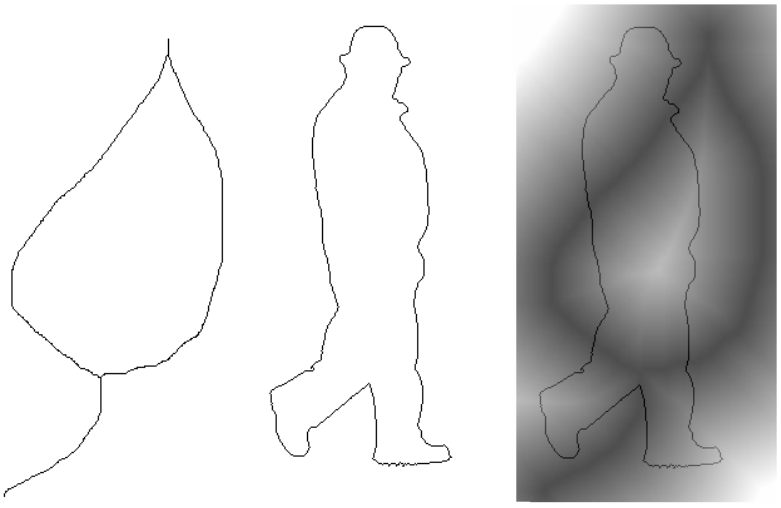


Figura 4.2 – De la stânga la dreapta: conturul obiectului model (frunză); conturul obiectului necunoscut (pieton); pietonul este suprapus peste transformata distanță a frunzei pentru a calcula costul de similitudine.

4.3 Detalii de implementare

Citirea imaginii de intrare ca și o imagine cu niveluri de gri:

```
Mat img = imread("filename", IMREAD_GRAYSCALE);
```

Inițializarea imaginii TD cu imaginea de intrare:

```
Mat td = src.clone();
```

Accesarea pixelilor dintr-o vecinătate de tip 8:

```
int di[8] = {-1,-1,-1,0,0,1,1,1};
int dj[8] = {-1,0,1,-1,1,-1,0,1};
int weight[8] = {3,2,3,2,2,3,2,3};


for(int k=0; k<8; k++)
     uchar   pixel   =   img.at<uchar>(i+di[k],
j+dj[k]);
```

4.4 Activitate practică

1. Implementați Transformata Distanță Chamfer. Calculați și vizualizați imaginea TD pentru imaginile de intrare: *contour1.bmp, contour2.bmp, contour3.bmp*. Rezultatele trebuie să coincidă cu cele prezentate în text. Pixelii de obiect sunt negri iar fundalul este alb.
2. Calculați imaginea TD pentru *template.bmp*. Evaluați costul de similitudine între imaginea model și cele două obiecte necunoscute: *unknown_object1.bmp* – pieton, *unknown_object2.bmp* – frunză. Costul de similitudine este media valorilor din imaginea TD de pe pozițiile punctelor de contur ale obiectului necunoscut.
3. Calculați costul de similitudine prin inversarea rolurilor de obiect necunoscut și obiect model.
4. Translatați obiectul necunoscut astfel încât centrul său să corespundă cu centrul obiectului model și recalculați costurile de potrivire.
5. Opțional, implementați Transformata Distanță Euclidiană adevărată. De ce este diferită față de Transformata Distanță Chamfer?

4.5 Exemple de rezultate

Exemple de rezultate de imagine TD folosind metoda Chamfer și matricea de ponderi sugerată.

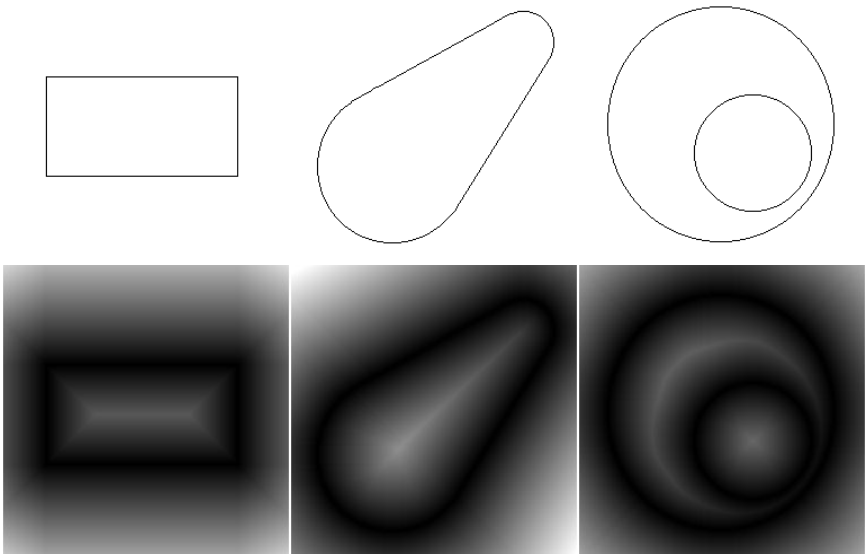


Figura 4.3 – Rândul de sus: imagini binare; rândul de jos: transformata distanță Chamfer

4.5 Referințe

[1] Wikipedia The Free Encyclopedia – *Distance Transform*, http://en.wikipedia.org/wiki/Distance_transform
[2] Compendium of Computer Vision – *Distance Transform*, http://homepages.inf.ed.ac.uk/rbf/HIPR2/distance.htm

5 Analiza statistică a datelor

5.1 Obiective

Scopul acestei lucrări este de a explora metodele de analiză statistică a datelor, folosite pentru clasificare și recunoaștere. Vom studia media, deviația standard si covarianța. Experimentele vor fi efectuate pe un set de imagini care conține fețe umane. Folosind matricea de covarianță, vom studia corelația dintre diferiți pixeli.

5.2 Fundamente teoretice

5.2.1 Definiții

În teoria probabilității, *spațiul rezultatelor* S reprezintă setul tuturor rezultatelor unui experiment. De exemplu, pentru experimentul de aruncare a unei monede, spațiul rezultatelor este S = {cap, pajură}. Un subset al spațiului rezultatelor se numește *eveniment*.

În multe cazuri, rezultatele sunt numerice, de exemplu atunci când corespund rezultatului măsurării cu un instrument, dar deseori acestea nu sunt numerice, dar pot fi asociate cu numere reale.

Având un experiment și un spațiu al rezultatelor, o *variabilă aleatorie* X este o funcție care atașează un număr real $X(\zeta)$ pentru fiecare posibil rezultat ζ din spațiul rezultatelor S al unui experiment aleatoriu, după cum se vede în Figura 5.1. Această funcție $X(\zeta)$ face o relaționare a tuturor posibilelor elemente din spațiul rezultatelor (eșantioanelor) cu domeniul numerelor reale (dreapta numerelor reale). Variabilele aleatorii pot fi:

- Discrete: numărul rezultat din aruncarea unui zar, numărul de capete obținut prin aruncarea de 2 ori a unei monede.
- Continue: greutatea unui individ.

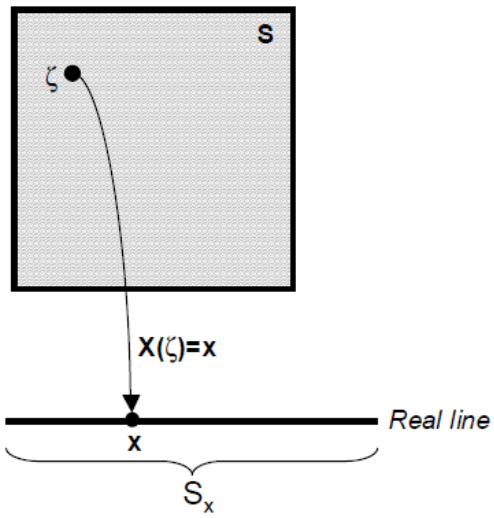


Figura 5.1 Exemplu de variabilă aleatorie.

Un *vector de variabile aleatorii X,* denumit și variabilă aleatorie *multivariată*, este un vector de variabile aleatorii asociate aceluiași experiment.

$$\boldsymbol{X} = [X_1, X_2, \dots, X_N]^T$$

5.2.2 Caracterizarea statistică a variabilelor aleatorii discrete

O variabilă aleatorie *X* se poate caracteriza prin probabilitățile valorilor pe care le poate lua. Pentru o variabilă aleatorie continuă, **funcția de densitate de probabilitate** (FDP) exprimă acest lucru.

Pentru o variabilă aleatorie discretă, **funcția de masă de probabilitate** (FMP) notată prin p_X exprimă acest lucru. Astfel, dacă x este orice valoare posibilă a lui X, funcția de masă de probabilitate $p_X(x)$ reprezintă probabilitatea evenimentului $\{X=x\}$:

$$p_X(x) = P(\{X = x\})$$

O proprietate importantă a funcției de masă de probabilitate este:

$$\sum_x p_X(x) = 1$$

unde x ia toate valorile posibile ale lui X.

Este deseori de dorit să sumarizăm funcția de masă de probabilitate printr-un singur număr. Astfel se pot calcula următoarele cantități:

1. *Media* reprezintă o medie ponderată de probabilități a posibilelor valori ale lui X

$$E[X] = \mu = \sum_x x p_X(x)$$

Un caz particular este acela al unei variabile distribuite uniform, unde probabilitățile sunt egale (*1/n*) pentru fiecare valoare a acesteia (în total X poate lua *n* valori). Un exemplu ar fi cel de aruncare a unui zar, iar variabila distribuită uniform este numărul de pe fața zarului. Astfel media se reduce la:

$$E[X] = \mu = \frac{1}{n}\sum_x x$$

2. *Varianța* (σ^2)sau dispersia reprezintă "împrăștierea" in jurul mediei:

$$VAR[X] = E[(X - E[X])^2] = \sum_x (x - \mu)^2 p_X(x)$$

3. *Deviația standard* (σ) este rădăcina pătrată a variației, se exprimă in aceleași unități ca variabila aleatorie:

$$STD[X] = VAR[X]^{1/2}$$

5.2.3 Caracterizarea statistică a vectorilor aleatorii discreți

Putem descrie parțial un vector aleatoriu prin următoarele valori:

1. *Vectorul mediu*:

$$E[\boldsymbol{X}] = \big[E[X_1], E[X_2], \dots, E[X_N]\big] = [\mu_1, \mu_2, \dots, \mu_N] = \boldsymbol{\mu}$$

2. *Matricea de covarianță*:

$$COV[\boldsymbol{X}] = \boldsymbol{\Sigma} = E[(\boldsymbol{X} - \boldsymbol{\mu})(\boldsymbol{X} - \boldsymbol{\mu})^T]$$

$$COV[\boldsymbol{X}] = \begin{bmatrix} VAR(X_1) & \dots & COV[(X_1, X_N)] \\ \dots & \dots & \dots \\ COV[(X_N, X_1)] & \dots & VAR(X_N) \end{bmatrix}$$

$$= \begin{bmatrix} E[(X_1 - \mu_1)(X_1 - \mu_1)^T] & \dots & E[(X_1 - \mu_1)(X_N - \mu_N)^T] \\ \dots & \dots & \dots \\ E[(X_N - \mu_N)(X_1 - \mu_1)^T] & \dots & E[(X_N - \mu_N)(X_N - \mu_N)^T] \end{bmatrix}$$

$$= \begin{bmatrix} \sigma_1^2 & \dots & c_{1N} \\ \dots & \dots & \dots \\ c_{N1} & \dots & \sigma_N^2 \end{bmatrix}$$

Pe diagonala matricei de covarianță avem varianța unei variabile X_i din vectorul aleatoriu X unde $i \in [1, N]$, iar pe celelalte poziții din matrice avem covarianța dintre două perechi de variabile (X_i, X_k) din vectorul aleatoriu.

Definim covarianța ca:

$$COV[X_i, X_k] = \sum_{x_i} \sum_{x_k} (x_i - \mu_i)(x_k - \mu_k) p_{X_i, X_k}(x_i, x_k)$$

Unde $p_{X_i, X_k}(x_i, x_k)$ este *funcția de masă de probabilitate comună* a variabilelor aleatorii X_i și X_k.

Matricea de covarianță indică tendința fiecărei perechi de variabile aleatorii să varieze împreună, sau să co-varieze.

Covarianța are câteva proprietăți importante:
- Dacă X_i și X_k cresc împreună, atunci $c_{ik} > 0$
- Dacă X_i tinde sa descrească atunci când X_k creste, atunci $c_{ik} < 0$
- Dacă X_i și X_k sunt necorelate, atunci $c_{ik} = 0$
- $|c_{ij}| < \sigma_i \sigma_j$, unde σ_i este deviația standard a lui X_i
- $c_{ii} = VAR[X_i]$
- $c_{ij} = c_{ji}$

Termenii matricei de covarianță pot fi scrișI ca:

$$c_{ik} = E[(X_i - \mu_i)(X_k - \mu_k)]$$
$$c_{ii} = \sigma_i^2$$
$$c_{ik} = \rho_{ik}\sigma_i\sigma_k$$

unde ρ_{ik} este numit **coeficientul de corelație Pearson** .

Figurile următoare prezintă **graficele de corelație** dintre variabilele X_i si X_k.

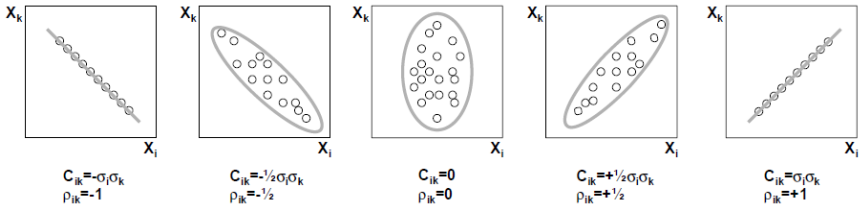


Figura 5.2 – Grafice de corelație între două variabile aleatorii X_i si X_k. De la stânga la dreapta: corelație negativă puternică, corelație negativă medie, nicio corelație, corelație pozitivă medie și corelație pozitivă puternică.

Coeficientul de corelație ρ_{ik} dintre două variabile reprezintă de fapt covarianța normalizată. Acesta ia valori între [-1,1]. Cu cât valoarea este mai apropiată de -1 sau 1, cu atât corelația este mai puternică. Covarianța se folosește atunci când scalele celor două variabile sunt la fel, iar coeficientul de corelație atunci când scalele sunt diferite.

5.3 Aspecte practice

Se dau p imagini, fiecare imagine conține o față umană (p = 400), ca în imaginile de mai jos din setul de date MIT CBCL FACE [1].

Se formează matricea de trăsături ***I*** care va conține intensitățile din toate imaginile de intrare. ***I*** are dimensiunea $p \times N$, unde p este numărul de imagini și *N* este numărul de pixeli din imagine. Rândul k conține toți pixelii din imaginea k rearanjați rând după rând în următorul mod:

$$\begin{bmatrix} A_{00} & A_{01} & A_{02} \\ A_{10} & A_{11} & A_{12} \\ A_{20} & A_{21} & A_{22} \end{bmatrix} \rightarrow [A_{00}, A_{01}, A_{02}, A_{10}, A_{11}, A_{12}, A_{20}, A_{21}, A_{22}]$$

Fiecare imagine din set are dimensiunea *N*=19x19 pixeli. Interpretarea matricei ***I*** este că fiecare rând conține un eșantion al variabilei aleatorii *N* dimensionale ***X***, care urmărește distribuția setului de date.
Obiectivul este calculul matricei de covarianță pentru un set dat de imagini și observarea modului în care variază împreună diferitele trăsături.

Valoarea medie a unei trăsături de la poziția *i* in imagine este:

$$\mu_i = \frac{1}{p} \sum_{k=1}^{p} I_{ki}$$

Unde I_{ki} reprezintă valoarea trăsăturii *i* din imaginea *k*.

Deviația standard a trăsăturii *i* este:

$$\sigma_i = \sqrt{\frac{1}{p} \sum_{k=1}^{p} (I_{ki} - \mu_i)^2}$$

Elementele matricei de covarianță c_{ij} pot fi calculate ca:

$$c_{ij} = \frac{1}{p} \sum_{k=1}^{p} (I_{ki} - \mu_i)(I_{kj} - \mu_j)$$

Iar coeficientul de corelație este:

$$\rho_{ij} = \frac{c_{ij}}{\sigma_i \sigma_j}$$

Se observă că $c_{ii} = \sigma_i^2$ și $\rho_{ii} = 1$.

5.4 Activitate practică

1. Încărcați cele 400 de imagini și stocați valorile de intensitate din fiecare imagine ca și rânduri in matricea de trăsături ***I***. Secțiunea de cod exemplu care încarcă setul de imagini este:

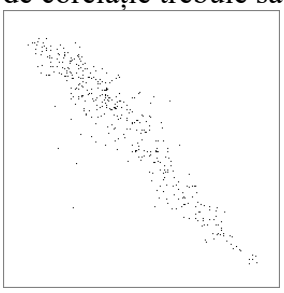
```
char folder[256] = "faces";
char fname[256];
for(int i=1; i<=400; i++){
      sprintf(fname,"%s/face%05d.bmp", folder, i);
      Mat img = imread(fname, IMREAD_GRAYSCALE);
}
```

2. Calculați vectorul cu valorile medii si salvați-l într-un fișier text de tip CSV (comma separated values). Se scriu componentele cu virgule între ele și se salvează într-un fișier text cu extensia CSV. Acest tip de fișier poate fi deschis cu Microsoft Excel sub forma unui tabel.

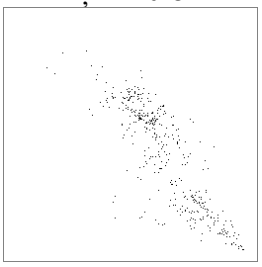
3. Calculați matricea de covarianță si salvați-o într-un fișier CSV.

4. Calculați matricea coeficienților de corelație și salvați-o într-un fișier CSV.

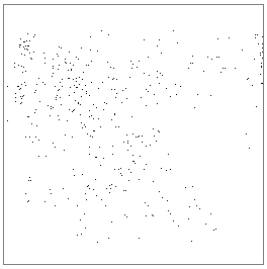
5. Afișați coeficientul de corelație si graficul de corelație pentru următoarele poziții (linie, coloana). Graficul de corelație este o imagine albă de dimensiune 256x256 care conține puncte negre la fiecare poziție (I_{kj}, I_{ki}), unde $k=1:p$ iar i și j au fost fixate și reprezintă poziții liniarizate din imagine.
    a. (5,4) și (5,14). Aceste puncte corespund unor pixeli aparținând ochiului stâng și drept. În acest exemplu, $i = 5 * 19 + 4$, iar $j = 5 * 19 + 14$. Rezultatul trebuie să fie asemănător cu cel din figura de mai jos, și coeficientul de corelație trebuie sa fie ~ 0.94.



b. (10,3) și (9,15). Aceste puncte corespund pixelilor de pe obrazul stâng și obrazul drept. Rezultatul trebuie să arate ca în figura de mai jos, cu un coeficient de corelație ~ 0.84.



c. (5,4) și (18,0). Aceste puncte corespund pixelilor care aparțin ochiului stâng și coltul din stânga jos al imaginii – deci puncte necorelate. Rezultatul ar trebui să arate ca în figura de mai jos, având coeficientul de corelație ~ 0.07.



6. Afișați graficul funcției de densitate de probabilitate unidimensională pentru o trăsătură aleasă. Formula funcției de densitate gaussiană este:

$$f_x(x) = \frac{1}{\sqrt{2\pi}\sigma} exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

unde μ este valoarea medie și σ este deviația standard pentru trăsătura selectată. Se normalizează valorile astfel încât maximul sa fie egal cu înălțimea imaginii.

7. Opțional, afișați densitatea de probabilitate 2D sub forma unei imagini cu niveluri de gri pentru două trăsături. Forma funcției de densitate gaussiană este:

$$p(x_i, x_j) = \frac{1}{2\pi\sqrt{\det(C_{ij})}} exp\left(-0.5\left(\left[x_i - \mu_i, x_j - \mu_j\right] C_{ij}^{-1} \begin{bmatrix} x_i - \mu_i \\ x_j - \mu_j \end{bmatrix}\right)\right)$$

unde μ_i este valoarea medie pentru trăsătura i iar C_{ij} este matricea de covarianță pentru trăsăturile *i* și *j*. Se normalizează valorile pentru a obține valori în intervalul 0:255.

5.5 Exemple

1. Fie experimentul de aruncare independentă a unei monede de două ori și fie X numărul de capete obținute. Spațiul rezultatelor este in acest caz S = {0, 1, 2}. Funcția de masă de probabilitate a lui X este:

$$p_X(x) = \begin{cases} \frac{1}{4}, & dacă\ x = 0\ sau\ x = 2 \\ \frac{1}{2}, & dacă\ x = 1 \end{cases}$$

Media lui X este:

$$E[X] = 0 \cdot \frac{1}{4} + 1 \cdot \frac{1}{2} + 2 \cdot \frac{1}{4} = 1$$

Varianța lui X este:

$$VAR[X] = (0-1)^2 \cdot \frac{1}{4} + (1-1)^2 \cdot \frac{1}{2} + (2-1)^2 \cdot \frac{1}{4} = \frac{1}{2}$$

2. Considerăm experimentul de aruncare a unui zar, pentru care avem probabilități egale de apariție a fiecărei fețe (1/6). Fie vectorul de variabile aleatorii [X,Y]. Variabila aleatorie X este egală cu 1 dacă numărul de pe față este par (2, 4, 6) și 0 altfel:

$$X = \begin{cases} 1, & dacă\ număr\ par \\ 0, & altfel \end{cases}$$

Variabila aleatorie Y este egală cu 1 dacă numărul de pe față este prim (2, 3, 5) și 0 altfel:

$$Y = \begin{cases} 1, & dacă\ număr\ prim \\ 0, & altfel \end{cases}$$

Funcția de masă de probabilitate a lui X este:

$$p_X(x) = \begin{cases} \frac{3}{6} = \frac{1}{2}, & dacă\ x = 1 \\ \frac{3}{6} = \frac{1}{2}, & dacă\ x = 0 \end{cases}$$

Media lui X este:

$$E[X] = 0 \cdot \frac{1}{2} + 1 \cdot \frac{1}{2} = \frac{1}{2}$$

Varianța lui X este:

$$\text{VAR}[X] = \left(0 - \tfrac{1}{2}\right)^2 \cdot \tfrac{1}{2} + \left(1 - \tfrac{1}{2}\right)^2 \cdot \tfrac{1}{2} = \tfrac{1}{8} + \tfrac{1}{8} = \tfrac{1}{4}$$

Funcția de masă de probabilitate a lui Y este:

$$p_Y(y) = \begin{cases} \frac{3}{6} = \frac{1}{2}, & dacă\ y = 1 \\ \frac{3}{6} = \frac{1}{2}, & dacă\ y = 0 \end{cases}$$

Media lui Y este:

$$E[Y] = 0 \cdot \frac{1}{2} + 1 \cdot \frac{1}{2} = \frac{1}{2}$$

Varianța lui Y este:

$$\text{VAR}[Y] = (0 - \tfrac{1}{2})^2 \cdot \tfrac{1}{2} + \left(1 - \tfrac{1}{2}\right)^2 \cdot \tfrac{1}{2} = \tfrac{1}{8} + \tfrac{1}{8} = \tfrac{1}{4}$$

Funcția de masă de probabilitate comună a lui X și Y definește probabilitățile pentru fiecare pereche de valori ale lui X și Y:

(X = 0, Y=0), (X=0, Y=1), (X=1, Y=0), (X=1, Y=1)

Vom calcula *funcția de masă de probabilitate comună* a variabilelor X și Y astfel:

$$p_{X,Y}(x,y) = \begin{cases} \frac{1}{6}, & dacă\ x = 0\ şi\ y = 0 \\ \frac{1}{6}, & dacă\ x = 1\ şi\ y = 1 \\ \frac{2}{6}, & dacă\ x = 0\ şi\ y = 1 \\ \frac{2}{6}, & dacă\ x = 1\ şi\ y = 0 \end{cases}$$

Putem calcula covarianța dintre X și Y:

$$COV[X_i, X_k] = \sum_{x_i} \sum_{x_k} (x_i - \mu_i)(x_k - \mu_k) p_{X_i,X_k}(x_i, x_k)$$

Matricea de covarianță este:

	X	**Y**
X	VAR(X) = 1/4	COV(X,Y)= –1/12
Y	COV(Y,X)= –1/12	VAR(Y) = 1/4

$$\begin{aligned} COV[X,Y] &= COV[Y,X] \\ &= \left(0 - \frac{1}{2}\right) * \left(0 - \frac{1}{2}\right) * \frac{1}{6} + \left(0 - \frac{1}{2}\right) * \left(1 - \frac{1}{2}\right) * \frac{2}{6} \\ &+ \left(1 - \frac{1}{2}\right) * \left(0 - \frac{1}{2}\right) * \frac{2}{6} + \left(1 - \frac{1}{2}\right) * \left(1 - \frac{1}{2}\right) * \frac{1}{6} \\ &= -\frac{2}{24} \end{aligned}$$

5.6 Referințe

[1] MIT CBCL FACE dataset
http://www.ai.mit.edu/courses/6.899/lectures/faces.tar.gz

6 Analiza Componentelor Principale

6.1 Obiective

În această lucrare de laborator se descrie metoda de Analiză a Componentelor Principale (*Principal Component Analysis* – PCA). Această metodă se utilizează pentru reducerea dimensionalității, compresia și vizualizarea datelor. Pentru realizarea acestei lucrări de laborator este necesară o bibliotecă care să calculeze valorile și vectorii proprii ale unei matrice (descompunerea în valori proprii).

6.2 Fundamente teoretice

Se consideră un set de puncte de date într-un spațiu de dimensiune mare (ND). Fiecare vector reprezintă trăsăturile unui exemplu de antrenare. Scopul acestei metode este reducerea dimensionalității punctelor la o dimensiune mai mică KD în așa fel încât să se păstreze cât mai multă informație cu putință. Ideea PCA este de a găsi principalele K axe de variație, astfel încât după proiecția datelor într-un spațiu mai redus, varianța datelor proiectate sa fie maximizată.

Inițial vom considera un exemplu bidimensional: se afișează datele colectate despre cât de mult apreciază anumite persoane niște activități și aptitudinea lor în domeniul respectiv. Figura 6.1 ilustrează un exemplu simplificat [2].

Să analizăm cei doi vectori $\boldsymbol{u}_1$ și $\boldsymbol{u}_2$. Dacă se proiectează punctele 2D pe vectorul $\boldsymbol{u}_2$ se obțin valori scalare cu o împrăștiere redusă (deviație standard mică). În schimb, dacă se proiectează punctele pe $\boldsymbol{u}_1$ punctele sunt mult mai împrăștiate. Dacă ar trebui să reducem datele la o singură dimensiune, atunci ar fi preferabil să le proiectăm pe $\boldsymbol{u}_1$ întrucât datele sunt mai ușor separabile iar varianța este mai mare.

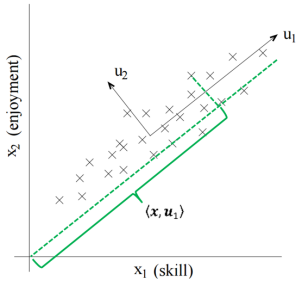


Figura 6.1 – Proiecția punctului x pe vectorul u_1

Exprimat într-un mod mai formal, fiecare punct bidimensional poate fi scris ca:

$$\boldsymbol{x} = \langle \boldsymbol{x}, \boldsymbol{u}_1 \rangle \, \boldsymbol{u}_1 / \|\boldsymbol{u}_1\| + \langle \boldsymbol{x}, \boldsymbol{u}_2 \rangle \, \boldsymbol{u}_2 / \|\boldsymbol{u}_2\|$$

În ecuația de mai sus punctul $\boldsymbol{x}$ a fost proiectat pe fiecare vector și apoi rezultatele obținute au fost însumate. Produsul scalar $\langle \boldsymbol{x}, \boldsymbol{u}_i \rangle$ definește magnitudinea proiecției și trebuie normalizat cu norma vectorului $\|\boldsymbol{u}_i\|$; cei doi vectori definesc direcțiile. Această exprimare este posibilă deoarece $\boldsymbol{u}_1$ și $\boldsymbol{u}_2$ sunt vectori perpendiculari. Dacă se pune condiția ca cei doi vectori să fie vectori unitate, atunci termenul de normalizare dispare. Pentru mai multe exemple de proiecţie vedeți [4]. Ideea principală a reducerii dimensionalității datelor este să se utilizeze cele mai mari proiecții. Întrucât proiecțiile pe $\boldsymbol{u}_2$ vor fi mai mici, $\boldsymbol{x}$ se poate aproxima folosind doar primul termen:

$$\tilde{\boldsymbol{x}}_{\mathbf{1}} = \langle \boldsymbol{x}, \boldsymbol{u}_1 \rangle \, \boldsymbol{u}_1 / \|\boldsymbol{u}_1\|$$

În general, fiind dată o bază ortonormală a unui spațiu vectorial cu *d* dimensiuni $\boldsymbol{B}$ cu vectorii de bază $\boldsymbol{b}_i$, orice vector se poate scrie ca:

$$\boldsymbol{x} = \sum_{i=1}^{d} \langle \boldsymbol{x}, \boldsymbol{b}_i \rangle \boldsymbol{b}_i = \sum_{i=1}^{d} (\boldsymbol{x}^T \boldsymbol{b}_i) \, \boldsymbol{b}_i$$

Problema revine acum să determinăm vectorii de bază pe care se vor realiza proiecțiile. Întrucât scopul principal este maximizarea varianței dintre punctele rezultate în urma transformării, matricea de covarianță ne poate oferi informațiile necesare. Covarianța dintre două trăsături este definită ca:

$$C(x_i, x_j) = \frac{1}{n-1} \sum_{k=1}^{n} (X_{ki} - \mu_i)(X_{kj} - \mu_j)$$

unde μ_i este media trăsăturii *i*. Matricea de covarianță stochează covariațele pentru toate perechile de trăsături. Se poate demonstra că matricea de covarianță poate fi exprimată ca un simplu produs de matrice:

$$C = \frac{1}{n-1} (X - \boldsymbol{\mu} \mathbf{1}_{1xn})^T (X - \boldsymbol{\mu} \mathbf{1}_{1xn})$$

unde $\boldsymbol{\mu}$ este un vector care conține valorile medii ale trăsăturilor și $\mathbf{1}_{1xn}$ este un vector rând ce conține doar valori de 1. Dacă eliminăm media din datele de intrare, într-un pas de preprocesare, ecuația se simplifică și mai mult:

$$C = \frac{1}{n-1} X^T X$$

Pasul următor este găsirea axelor de-a lungul cărora covarianța este maximă. Descompunerea în valori proprii ale unei matrice ne furnizează aceste informații. Astfel, vectorul propriu corespunzător celei mai mare valori proprii va reprezenta prima axă principală și așa mai departe.

Intuitiv, (aproape) orice matrice poate fi vizualizată ca o rotație urmată de o scalare de-a lungul axelor și rotația inversă. Descompunerea în vectori și valorilor proprii calculează această descompunere a matricei de covarianță:

$$C = Q\Lambda Q^T = \sum_{i=1}^{d} \lambda_i Q_i Q_i^T$$

unde *Q* este o matrice de rotație de dimensiune *dxd* (ortonormală) și *Λ* este o matrice diagonală ale cărei elemente reprezintă scalarea de-a lungul fiecărei axe. Elementele se numesc valori proprii și fiecare coloană corespunzătoare din *Q* este vectorul propriu corespunzător. Deoarece scopul principal este menținerea proiecțiilor cu varianță maximă, valorile proprii se ordonează descrescător în funcție de magnitudinea lor și se aleg primele *k* valori proprii. Astfel *C* poate fi aproximat ca:

$$\tilde{C}_k = Q_{1:k}\Lambda_{1:k}Q_{1:k}^T = \sum_{i=1}^{k} \lambda_i Q_i Q_i^T$$

unde $Q_{1:k}$ este o matrice de dimensiune *dxk* cu primii *k* vectori proprii și $\Lambda_{1:k}$ este o matrice diagonală de dimensiune *kxk* ce conține primele *k* valori proprii. Dacă *k* este egal cu *d* se obține matricea originală și, pe măsură ce valoarea lui *k* scade, se obțin aproximări tot mai grosiere ale lui *C*.

Astfel am determinat axele de-a lungul cărora varianța proiecțiilor este maximizată. În cazul general un vector poate fi aproximat cu *k* vectori astfel:

$$\tilde{\boldsymbol{x}}_{\boldsymbol{k}} = \sum_{i=1}^{k} \langle \boldsymbol{x}, Q_i \rangle Q_i = \sum_{i=1}^{k} (\boldsymbol{x}^T Q_i)\, Q_i$$

unde Q_i este coloana *i* a matricei de rotație *Q*.

Coeficienţii PCA pot fi calculaţi ca:

$$X_{coef} = XQ$$

Proiecția PCA (de la *d* la *k* dimensiuni, $k < d$) poate fi calculată ca:

$$X_k = XQ_{1:k}$$

unde $Q_{1:k}$ este matricea formată din primele *k* coloane din *Q*.

Aproximarea PCA (reconstrucția PCA de la *k* la *d* dimensiuni, $k < d$**)** poate fi calculată pentru toți vectorii de intrare simultan (dacă ei sunt stocați ca rânduri în *X*) utilizând formula:

$$\widetilde{\boldsymbol{X}}_{\boldsymbol{k}} = \sum_{i=1}^{k} \boldsymbol{X}\boldsymbol{Q}_i\boldsymbol{Q}_i^t = \sum_{i=1}^{k} \boldsymbol{X}_{\boldsymbol{coef}_i}\boldsymbol{Q}_i^t = \boldsymbol{X}\boldsymbol{Q}_{\boldsymbol{1:k}}\boldsymbol{Q}_{\boldsymbol{1:k}}^{\boldsymbol{T}}$$

unde $Q_{1:k}$ este matricea formată din primele *k* coloane din *Q*. Este important să se facă distincția între aproximare și coeficienți: aproximarea este suma coeficienților înmulțite cu componentele principale.

În finalul acestei prezentări teoretice vom trece în revistă mai multe exemple în care PCA se poate aplica cu succes:

- Reducerea dimensionalității trăsăturilor: în unele cazuri, vectori de trăsături cu o dimensionalitate mare pot să încetinească procesul de predicție
- Vizualizarea datelor – datele pot fi analizate în 3D sau în 2D; pentru date cu o dimensionalitate mai mare este necesară proiecția datelor;
- Aproximarea vectorilor de date;
- Detecția trăsăturilor redundante și a dependențelor liniare dintre trăsături;
- Reducerea zgomotului – dacă zgomotul din date are o varianță mai mică decât datele, adică raportul dintre semnal și zgomot (SNR) este mare, atunci PCA elimină zgomotul din datele de intrare.

6.3 Detalii de implementare

Declararea și alocarea unei matrice de dimensiune *n* x *d* cu valori flotante exprimate în dublă precizie:

```
Mat X(n,d,CV_64FC1);
```

Calculul matricei de covarianță după ce mediile au fost scăzute din valorile de intrare:

```
Mat C = X.t()*X/(n-1);
```

Pentru a calcula descompunerea în valori proprii, *Lambda* va conține valorile proprii și Q va conține vectorii proprii. Este necesară transpunerea deoarece X conține datele de intrare de-a lungul rândurilor.

```
Mat Lambda, Q;
eigen(C, Lambda, Q);
Q = Q.t();
```

Produsul scalar este implementat ca o simplă înmulțire. Atenție, datorită faptului că indexarea începe de la 0, primul rând este *row*(0). Produsul scalar dintre rândul *i* din X și coloana *i* din Q este dat de:
`Mat prod = X.row(i)*Q.col(i)`

6.4 Activitate practică

1. Deschideți fișierul de intrare și citiți punctele de date. Pe prima linie este stocat numărul de puncte *n* și dimensionalitatea datelor de intrare *d.* Liniile următoare din fișier conțin câte un punct cu *d* coordonate. Calculați vectorul cu valorile medii și scădeți-l din punctele de intrare.
2. Calculați matricea de covarianță ca un produs de matrice.
3. Efectuați descompunerea în valori proprii a matricei de covarianță apelând funcţia din bibliotecă.
4. Afișați valorile proprii.
5. Calculați coeficienții PCA și aproximarea $\tilde{X}_k$ de ordinul *k* (folosind primele *k* valori proprii) pentru datele de intrare.
6. Calculați valoarea medie a diferenței absolute dintre punctele originale și aproximarea lor utilizând primele *k* componente principale.
7. Găsiți minimele și maximele pe coloanele matricei de coeficienți.
8. Pentru datele de intrare din fișierul *pca2d.txt*, afișați coeficienţii PCA din primele două coloane ca puncte negre 2D pe fundal alb. Pentru a obține coordonate pozitive scădeți valorile minime.
9. Pentru datele de intrare din fișierul *pca3d.txt*, afişaţi coeficienţii PCA din primele trei coloane sub forma unei imagini cu niveluri de gri. Utilizați primele 2 componente ca și coordonatele *x* și *y*, iar cea de-a treia valoare ca intensitate în punctul (*x*, *y*). Pentru a obține

coordonate pozitive trebuie să scădeți valoarea minimă din primele două coordonate. Normalizați a treia componentă astfel încât să ia valori între 0:255.

10. Determinați automat numărul de componente principale k care trebuie să fie păstrate astfel încât să se reţină un anumit procent din varianța inițială. De exemplu, găsiți valoarea lui k pentru care aproximarea de ordinul k reține 99% din varianța inițială. Procentajul variației păstrate este dat de $\sum_{i=1}^{k} \lambda_i / \sum_{i=1}^{d} \lambda_i$.

6.5 Exemple de rezultate

Pentru *pca2d*

- Prima valoare proprie este aproximativ 8090.21
- Eroarea medie absolută folosind o singură componentă: 22.43

Pentru *pca3d*

- Prima valoare proprie este aproximativ 5462.33
- Eroarea medie absolută folosind o singură componentă: 14.50

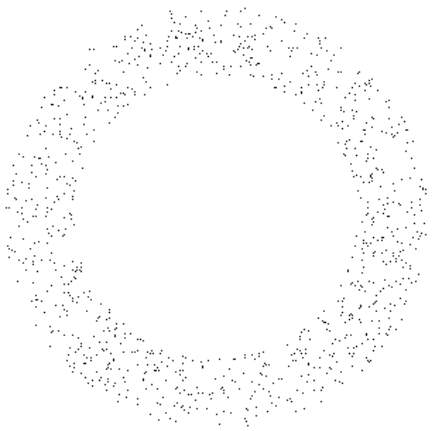


Figura 6.2. Vizualizarea punctelor ce rezultă după aplicarea metodei PCA pe datele din fișierul *pca2d.txt*

6.6 Referințe

[1] Wikipedia article PCA - https://en.wikipedia.org/wiki/Principal_component_analysis
[2] Stanford CS229 Machine Learning course notes -

https://cs229.stanford.edu/main_notes.pdf
[3] Lindsay Smith - PCA tutorial -
http://faculty.iiit.ac.in/~mkrishna/PrincipalComponents.pdf
[4] PCA in R (animation of projection) -
https://poissonisfish.wordpress.com/2017/01/23/principal-component-analysis-in-r/

7 Algoritmul de grupare K-means

7.1 Obiective

În această lucrare de laborator se va rezolva problema grupării unui set de puncte (*clustering*). Această sarcină face parte din învățarea automată nesupervizată, în sensul că etichetele/clasele punctelor nu sunt cunoscute și nici nu sunt necesare în procesul de învățare. Prin utilizarea unor metode adecvate se va identifica structura datelor și punctele similare vor fi grupate împreună.

7.2 Fundamente teoretice

Scopul metodelor de grupare (*clustering*) este de a partiţiona o mulţime de obiecte în grupuri diferite, unde instanţele dintr-un grup sunt similare într-un anumit sens. Gruparea este folosită în multe domenii precum: învăţare computerizată, sisteme de recunoaştere a formelor, analiza imaginilor, bioinformatică, compresie, grafică, etc.

Algoritmul k-means primește ca valori de intrare o listă de puncte $X = \{x_i,\ i = 1{:}n\}$. Fiecare punct este d-dimensional $x_i = (x_{i1}, x_{i2}, \dots, x_{id})$. Obiectivul metodei k-means este gruparea punctelor în K mulțimi notate cu $S = \{S_k |\ k = 1{:}K\}$. Centrul fiecărei mulțimi (grup) S_k este notat cu m_k. Gruparea datelor trebuie realizată astfel încât să fie minimizată funcția obiectiv:

$$J(X,S) = \sum_{k=1}^{K} \sum_{x \in S_k} dist(x, m_k)$$

unde $dist(.,.)$ este distanţa Euclidiană în spaţiul d-dimensional:

$$dist(x,y) = \sqrt{\sum_{i=1}^{d} (x_i - y_i)^2}$$

Aceasta este o problemă NP-hard, dar există anumite soluții aproximative prin care se obțin rezultate bune. Metoda lui Lloyd propune separarea problemei în două părți. Dacă se cunoaște modul în care punctele sunt partiționate se pot calcula centrele, dar nu se pot cunoaște partițiile dacă centrele grupurilor nu sunt cunoscute. Ideea este să se înceapă cu o mulțime aleatorie a centrelor grupurilor și apoi

centrele să fie schimbate în mai multe iterații. Este demonstrat că algoritmul va găsi o partiționare care corespunde unui minim local al funcției *J*, dar care nu este însă întotdeauna minimul global [2].

Fie *L* funcția de apartenență pentru fiecare punct, deci $L(i) \in 1{:}K, i = 1{:}n$. Această funcţie returnează grupul din care face parte al *i*-lea punct. Se începe prin selectarea aleatorie a centrelor grupurilor din punctele din setul de date: $m_k = x_{r_k}$, unde r_k este o valoarea întreagă uniform distribuită între 1 şi *n*. Pentru a asigura convergența algoritmului se pot aplica tehnici mai complexe de inițializare. În [2] se definește metoda k-means++ bazată pe selectarea punctelor după o distribuție de probabilitate care penalizează punctele apropiate.
În continuare se aplică iterativ pașii de atribuire și de actualizare. Când apartenența punctelor la grupuri nu se mai modifică sau când se atinge un număr maxim de iterații, algoritmul se termină. Pașii metodei sunt prezentați în algoritmul următor:

Algoritmul K-means

Inițializare – Se selectează aleatoriu *K* centre din lista punctelor de intrare. Fie r_k o variabilă aleatorie, întreagă şi uniform distribuită în intervalul [1, *n*], atunci centrele inițiale sunt selectate ca:

$$m_k = x_{r_k}$$

Atribuire – Fiecare punct din lista de intrare este asociat cu centrul cel mai apropiat. Funcţia de apartenență va lua valoarea indexului celui mai apropiat centru:

$$L(i) = argmin_k dist(x_i, m_k)$$

Actualizare – Se recalculează centrele grupurilor pe baza funcției de apartenență. Noile centre ale grupurilor sunt calculate ca media punctelor din acel grup. În formula următoare se însumează toate punctele care aparțin grupului *k*, adică au funcția de apartenență $L(i) = k$.

$$m_k = \frac{\sum_{L(i)=k} x_i}{\sum_{L(i)=k} 1} = \frac{\sum_{x \in S_k} x}{|S_k|}$$

Condiția de terminare – Dacă nu apare nici o schimbare în funcția de apartenență, atunci algoritmul poate fi oprit deoarece calculele viitoare nu vor produce nici o schimbare în valorile centrelor. De asemenea, se poate limita numărul maxim de iterații ale algoritmului. Dacă nici una din condițiile de mai sus nu este îndeplinită, atunci algoritmul continuă cu pasul de atribuire.

7.3 Detalii de implementare

Generarea unei valori întregi aleatorie uniform distribuite din intervalul [*a, b*] (inclusiv):

```
#include <random>

default_random_engine gen;
uniform_int_distribution<int>  distribution(a,
b);
int rand_val = distribution(gen);
```

Crearea unei imagini color:

```
Mat img(height, width, CV_8UC3);
```

Construirea unui tablou de culori aleatorii pentru grupuri:

```
const int K = 3;
Vec3b colors[K];
for(int i = 0; i<K; i++)
     colors[i] = { (uchar)distribution(gen),
                   (uchar)distribution(gen),
                   (uchar)distribution(gen) };
```

Atribuirea unei culori *colors*[*k*] la poziția (*i*, *j*) în imagine:

```
img.at<Vec3b>(i,j) = colors[k];
```

7.4 Activitate practică

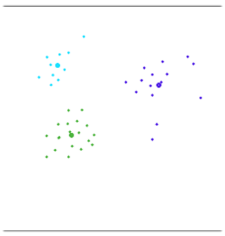
1. Implementaţi metoda de grupare K-means pe o listă de puncte *d* dimensionale. Aplicați iterativ algoritmul până când nu mai apare nici o modificare în funcția de apartenență sau până când numărul maxim de repetări ale algoritmului este atins. Numărul de grupuri *K* este specificat de utilizator în fiecare caz.
2. Aplicaţi K-means pe o mulţime de puncte din spaţiul 2D (imaginile *points*.bmp*) În acest caz *d=2*.
    a. Alegeţi culori în mod aleatoriu pentru grupuri şi coloraţi punctele în funcţie de apartenenţa lor.
    b. Pentru o vizualizare mai bună coloraţi şi vecinătatea punctelor.
    c. Desenaţi mozaicarea Voronoi corespunzătoare centrelor grupurilor obţinute. Aceasta presupune colorarea fiecărui

pixel din imagine (şi cei care aparţin fundalului) cu culoarea celui mai apropiat centru de grup.

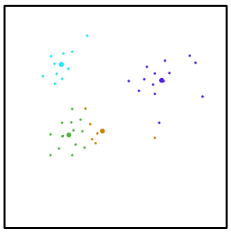
3. Aplicaţi K-means pe imagini cu niveluri de gri. În acest caz se grupează intensităţile care apar în imagine. Duplicatele se păstrează, adică de exemplu, dacă intensitatea 0 apare de 10 ori, vom avea 10 puncte egale cu 0. În acest caz *d=1*.
    a. Recoloraţi imaginea în funcţie de intensitatea medie a fiecărui grup.
4. Aplicaţi K-means pe imagini color. În acest caz se grupează culorile care apar în imagine, adică componentele R, G, B . În acest caz *d=3*.
    a. Recoloraţi imaginea în funcţie de culoarea medie a fiecărui grup.
5. Opţional, implementați metoda de inițializare k-means++ din [3].

7.5 Exemple de rezultate

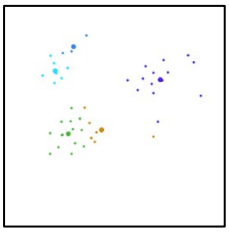
Pentru *d=2*, când algoritmul K-means este rulat pe un set de puncte 2D obţinem următoarele rezultate:



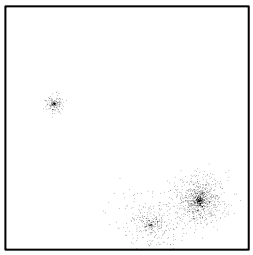
points2.bmp , K = 3



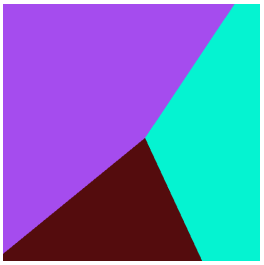
points2.bmp , K = 4



points2.bmp , K = 5



points4.bmp



points4.bmp - mozaicare Voronoi, K=3

Figura 7.1 – Punctele 2D și grupurile obținute cu diferite valori ale lui K. Exemplu de mozaicare Voronoi

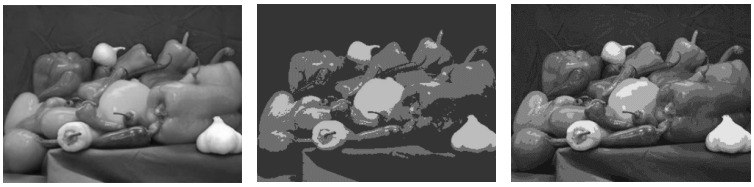


Figura 7.2 – Imagine de intensitate (stânga) și
rezultatele grupării cu K=3 (mijloc) și K=10 centre (dreapta)

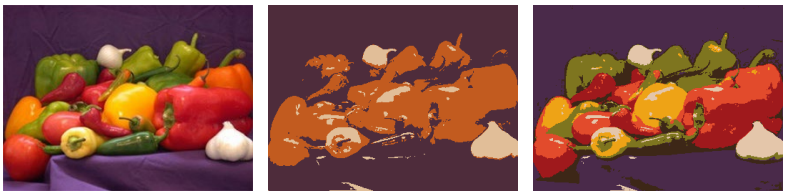


Figura 7.3 – Imagine color (stânga) și rezultatele grupării cu K=3 (mijloc) și K=10 centre (dreapta)

7.6 Referinţe

[1] Cluster analysis Wikipedia article - https://en.wikipedia.org/wiki/Cluster_analysis

[2] K-means Wikipedia article - https://en.wikipedia.org/wiki/K-means_clustering

[3] Arthur, David, and Sergei Vassilvitskii. "k-means++: The advantages of careful seeding." Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms. Society for Industrial and Applied Mathematics, 2007.

[4] P. Arbelaez, M. Maire, C. Fowlkes and J. Malik. „Contour Detection and Hierarchical Image Segmentation”, IEEE TPAMI, Vol. 33, No. 5, pp. 898-916, May 2011.

[5] Image segmentation dataset: https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/resources.html

8 Clasificatorul k-Nearest Neighbor

8.1 Obiective

Scopul acestei lucrări de laborator este introducerea clasificatorului "cei mai apropiaţi *K* vecini" (*k-Nearest Neighbor*) care, într-un sens, poate fi considerat clasificatorul cel mai elementar. Acesta va fi aplicat pe o problemă de clasificare cu mai multe clase.

8.2 Fundamente teoretice

8.2.1 Introducere

Un clasificator este definit ca o funcţie care returnează clasa unei instanţe. Instanţa sau exemplul este de obicei reprezentat prin vectorul de trăsături. Clasificatorul k-NN poate fi considerat cel mai simplu clasificator deoarece nu construieşte un model pentru setul de antrenare. Decizia se ia în funcţie de cei mai apropiaţi *K* vecini din setul de antrenare. Figura următoare ilustrează acest procedeu unde instanţa de test este pătratul albastru din centru înconjurat de instanţe etichetate din setul de antrenare. Interiorul cercului cuprinde 5 vecini pe baza cărora se va face clasificarea. Cercul are rază variabilă şi întotdeauna cuprinde *K* vecini.

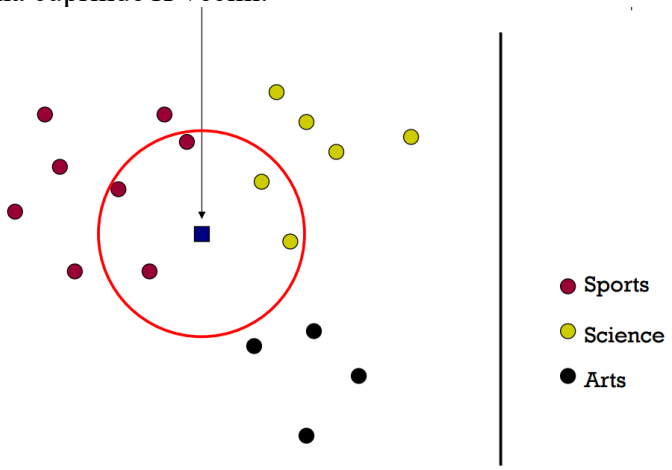


Figura 8.1 Exemplu de clasificator 5-NN pentru trei clase

Clasificatorul k-NN este un clasificator non-parametric, însemnând că nu construieşte un model pentru clasele pe care trebuie să le distingă. Se memorează setul de antrenare în întregime şi decizia asupra unei

instanţe de test se face online. k-NN poate fi categorisit ca şi o metodă de învăţare bazată pe instanţe sau învăţtare leneşă deoarece funcţia de decizie este aproximată doar local şi decizia asupra instanţei se amână la clasificare.

8.2.2 Algoritmul de clasificare

Fie setul de antrenare definit ca *X*, unde *X* este o matrice de dimensiune *n* x *d*. Fiecare linie din *X* conţine un vector de trăsături de dimensiune *d* denumit X_i, care corespunde unei instanţe de antrenare. Notăm cu $\boldsymbol{y}$ vectorul care conţine etichetele pentru clase. Dimensiunea lui $\boldsymbol{y}$ este de *n* x *1*, fiecare instanţă de antrenare având o clasă asociată. Elementele din $\boldsymbol{y}$ fac parte din mulţimea *{1,2,...,C}*, unde *C* este numărul de clase. Pentru o instanţă de test $\boldsymbol{x}$ se calculează distanţele dintre $\boldsymbol{x}$ şi fiecare exemplu de antrenare:

$$d_i = dist(x, X_i)$$

Distanţele sunt sortate în mod crescător şi cele mai apropiate *K* instanţe sunt luate în considerare. Fiecare instanţă votează pentru clasa lui care este cunoscută din $\boldsymbol{y}$. Instanţa de test va primi clasa cu cele mai multe voturi.

Fie **p** permutaţia care sortează distanţele în ordine crescătoare:

$$d_{p_1} < d_{p_2} < \cdots < d_{p_n}$$

Histograma de voturi de dimensiune *Cx1* este construită în următorul mod:

$$\boldsymbol{h} = \sum_{k=1}^{K} \mathbf{1}(y_{p_k})$$

unde $\mathbf{1}(y_{p_k})$ este un vector indicator de dimensiune *Cx1* care conţine 1 pe poziţia $\boldsymbol{y}_{p_k}$ şi 0 altundeva. Suma acumulează voturile de la cei mai apropiaţi *K* vecini. Clasa instanţei se alege ca:

$$c = argmax_i \, \boldsymbol{h}_i$$

Există mai multe versiuni ale algoritmului în funcţie de tipul de distanţă folosit şi metoda de votare. De exemplu, voturile pot fi ponderate cu inversa distanţei folosind formula:

$$\boldsymbol{h} = \sum_{k=1}^{K} \frac{\mathbf{1}(y_{p_k})}{1 + d_{p_k}}$$

unde s-a adăugat 1 la distanţă pentru a evita împărţirea la 0 şi pentru a obţine o pondere de 1 în cazul în care distanţa este egală cu 0.

Parametrul *K* controlează câţi vecini se iau în considerare. Dacă este egal cu *1* numai vecinul cel mai apropiat este considerat. Creşterea lui *K* reduce influenţa zgomotului dar îngreunează separarea claselor. În cazul extrem când *K=n*, setul întreg de antrenare este considerat. Dacă voturile nu sunt ponderate atunci instanţa de test este clasificată ca şi clasa cea mai frecventă. De multe ori *K* este ales să fie un număr impar pentru a rezolva situaţiile de egalitate dintre două clase. Pentru a alege valoarea potrivită pentru *K* se evaluează clasificatorul pe un set de validare şi se păstrează valoarea care duce la scorul cel mai bun (*hyperparameter optimization*).

Abordarea prezentată poate fi folosită şi pentru a realiza regresie dacă în loc de alegerea clasei se face o sumă ponderată a instanţelor de antrenare. Rata de eroare a clasificatorului k-NN tinde către eroarea ideală Bayes şi este mărginită de eroarea Bayes înmulţită cu 2 când numărul de instanţe tinde către infinit ($n \to \infty$).

8.2.3 Trăsături globale ale unei imagini

Imaginile color pot fi caracterizate printr-un vector global de trăsături cu scopul de a realiza clasificarea lor. Un vector de trăsături global are o dimensiune fixă şi descrie statistici globale despre imagine. De multe ori în aceste descrieri se pierde informaţia referitoare la aranjarea spaţială.

Histograma imaginii poate fi considerată o trăsătură globală pentru a descrie imaginea. Histograma se defineşte ca vectorul care conţine numărul de apariţii pentru fiecare nivel de intensitate. În acest caz vectorul are dimensiunea de 256 pentru o imagine de intensitate cu 8 biţi/pixel. În general, o histogramă cu *m* acumulatoare (eng. *bins*) conţine numărul de apariţii pentru intensităţile din fiecare acumulator. Se împarte intervalul de [0, 255] în *m* zone egale. De exemplu, pentru *m=8* acumulatoare, primul va conţine numărul de pixeli cu intensităţi cuprinse între [0, *256/m*); al doilea va conţine numărul de pixeli cu intensităţi între *32* and *63*; ş.a.m.d. Histograma unei imagini color se formează prin concatenarea histogramelor pe canalele individuale. Mărimea histogramei rezultate este de *3* x *m*.

8.2.4 Evaluarea clasificatorilor

Pentru a evalua performanţa clasificatorilor se folosesc mai multe metrici. **Matricea de confuzie** pentru un set etichetat se poate defini ca matricea care conține în fiecare celulă M_{ij} numărul de instanţe clasificate în clasa i şi care aparțin clasei j în realitate. Clasificatorul ideal atribuie la fiecare instanţă eticheta corectă şi deci asigură valori mari pe diagonala matricei de confuzie. În general valorile din matrice arată care clase sunt confundate între ele. De exemplu pentru un clasificator binar, matricea de confuzie este următoarea:

	Clasa reală: *Positive*	**Clasa reală: *Negative***
Clasa prezisă: *Positive*	**TP** (*True Positive*)	**FP** (*False Positive*)
Clasa prezisă: *Negative*	**FN** (*False Negative*)	**TN** (*True Negative*)

Figura 8.2 – Matricea de confuzie

Acurateţea unui clasificator pentru un set etichetat se defineşte ca procentajul de instanţe clasificate corect. Este măsura complementară erorii de clasificare. Nu oferă informaţie relevantă când clasele nu sunt echilibrate (sunt mai multe instanţe într-o clasă decât în cealaltă). Aceasta este o situaţie tipică, de exemplu un detector de pietoni trebuie să înveţe dintr-un set unde instanţele de fundal sunt mult mai numeroase. Un clasificator care returnează întotdeauna clasa mai frecventă poate să obţină o acurateţe ridicată. În acest caz trebuie să utilizăm alte metrici cum ar fi precizia pentru clasele individuale. Acurateţea se poate calcula din matricea de confuzie:

$$Acc = \frac{\sum_{i=1}^{C} M_{ii}}{\sum_{i=1}^{C} \sum_{j=1}^{C} M_{ij}}$$

8.3 Setul de date – recunoaşterea scenelor

Setul de date considerat conţine diferite scene. Există 6 clase: plajă, oraş, deşert, pădure, peisaj şi zăpadă. Imaginile pentru fiecare clasă sunt organizate în subdirectoare şi sunt numerotate cu numere de 6 cifre. Setul este neechilibrat din punctul de vedere al numărului de exemple pentru fiecare clasă și variază între 35 şi 277. Setul de antrenare este compus din 672 imagini, iar setul de testare conţine 85

de imagini. Mai jos sunt prezentate exemple reprezentative pentru fiecare clasă.

plajă oraş deşert pădure peisaj zăpadă

Figura 8.3 – Imagini reprezentative

8.4 Detalii de implementare

Sugestie pentru antetul funcţiei care calculează histograma (*hist* a fost alocată în prealabil):

```
void calcHist(Mat img, int nr_bins, int* hist)
```

Definirea denumirilor claselor:

```
const int nrclasses = 6;
char classes[nrclasses][10] =
      {"beach", "city", "desert", "forest", "landscape",
"snow"};
```

Se alocă memorie pentru matricea de trăsături și vectorul de etichete:

```
Mat X(nrinst, feature_dim, CV_32FC1);
Mat y(nrinst, 1, CV_8UC1);
```

Citirea imaginilor din clasa *c;* se calculează histograma şi se introduce ca şi o linie în *X*:

```
int c = 0, fileNr = 0, rowX = 0;
while(1){
      sprintf(fname, "train/%s/%06d.jpeg", classes[c],
fileNr++);
      Mat img = imread(fname);
      if (img.cols==0) break;

      calcHist(img, nr_bins, hist);

      for(int d=0; d<hist_size; d++)
            X.at<float>(rowX, d) = hist[d];
      y.at<uchar>(rowX) = c;
      rowX++;
}
```

Se alocă matricea de confuzie:

```
Mat C(nrclasses, nrclasses, CV_32FC1);
```

8.5 Activitate practică

1. Implementaţi o funcţie care extrage histograma dintr-o imagine color.
2. Citiţi toate imaginile din setul de antrenare. Calculaţi histograma generală cu *3* x *m* acumulatoare pentru fiecare imagine. Salvaţi histograma ca şi o linie în matricea de trăsături *X*. Salvaţi eticheta instanţei corespunzătoare pe aceeaşi linie în vectorul ***y*.**
3. Implementaţi clasificatorul k-NN având ca şi intrare o imagine şi valoarea lui *K*.
4. Evaluaţi clasificatorul pe setul de test prin calcularea matricei de confuzie şi a acurateţei.
5. Încercaţi să obţineţi rezultate cât mai bune folosind diferite valori pentru numărul de acumulatoare *m* şi pentru numărul de vecini *K*. Se poate obţine acurateţe de peste 65%.
6. Convertiţi imaginea de intrare într-un alt spaţiu de culoare (LUV sau HSV) înaintea calculării histogramei.
7. Opţional, încercaţi trăsături mai complexe (histograma pe regiuni) sau alte funcţii de distanţă (Manhattan distance, Euclidean ponderat).

8.6 Referinţe

[1] Wikipedia article - k-NN classifier
https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm
[2] Andrew Ng - Machine Learning: Nonparametric methods & Instance-based learning
http://www.cs.cmu.edu/~epxing/Class/10701-08s/Lecture/lecture2.pdf

9 Clasificatorul Bayes Naiv

9.1 Obiective

În această lucrare de laborator se prezintă clasificatorul Bayes Naiv. Se va aplica acest clasificator pe o problemă de recunoaştere a cifrelor scrise de mână.

9.2 Fundamente teoretice

Clasificatorul Bayes Naiv aplică regula Bayes şi presupune independenţa trăsăturilor pentru a calcula probabilităţile posterioare. Clasa cu probabilitate posterioară maximă va fi aleasă în momentul clasificării.

Regula lui Bayes se definește pentru două evenimente A și B independente și descrie probabilitatea condiționată al unui eveniment A dat de un alt eveniment B:

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

$P(A|B)$ se numește probabilitatea posterioară, iar $P(A)$ este probabilitatea a priori.

Datorită presupunerii de independenţă, clasificatorul poate lucra cu un număr arbitrar de trăsături. Pornim de la vectorul aleator cu d componente $\boldsymbol{x} = \{x_1, x_2, \dots, x_d\}$. Dorim să estimăm probabilitatea posterioară pentru clasele $C = \{c_1, c_2, \dots, c_J\}$. Într-un limbaj mai familiar, valorile din $\boldsymbol{x}$ sunt trăsăturile care determină clasa instanţei. Aplicând regula lui Bayes putem scrie:

$$p(c|x_1, x_2, \dots, x_d) \propto P(c)p(x_1, x_2, \dots, x_d|c)$$

unde $p(c|x_1, x_2, \dots, x_d)$ este probabilitatea posterioară, mai precis, probabilitatea clasei c dat fiind valorile trăsăturilor; $p(x_1, x_2, \dots, x_d|c)$ este funcţia de verosimilitate (*likelihood function*); iar $P(C)$ este probabilitatea a priori. Bayes Naiv presupune că trăsăturile

condiţionate de clasă sunt independente, deci putem să descompunem verosimilitatea într-un produs în următorul mod:

$$p(\boldsymbol{x}|c) = \prod_{k=1}^{d} p(x_k|c)$$

şi să rescriem probabilitatea posterioară ca:

$$p(c|\boldsymbol{x}) \propto P(C) \prod_{k=1}^{d} p(x_k|c)$$

Utilizând regula Bayes de mai sus putem să implementăm un clasificator. Acesta va returna clasa cu probabilitate posterioară maximă:

$$c^* = argmax_j \, p(c_j|\boldsymbol{x})$$

Deşi presupunerea de independenţă nu este în general valabilă aceasta simplifică problema de clasificare fiindcă permite calcularea probabilităţilor separat pentru fiecare clasă. În esenţă, se reduce problema estimării unei densităţi de probabilitate multidimensionale la mai multe probleme unidimensionale. Densitatea de probabilitate poate avea forme diferite precum: distribuţie normală, log-normală, gamma, Poisson sau variante discrete ale acestora.

9.3 Aspecte practice

9.3.1 Setul de date MNIST

Se utilizează un set de date standard pentru recunoaşterea cifrelor. Setul de date MNIST a fost asamblat de Yann LeCun din mai multe seturi. Partea de antrenare conţine 60000 imagini cu cifre scrise de mână de aproximativ 250 de persoane. Setul de test conţine 10000 instanţe. Distribuţia cifrelor este aproape uniformă. Pentru mai multe detalii accesaţi linkul [2]. Se utilizează distribuţii binomiale pentru a modela verosimilitatea. O distribuţie binomială specifică probabilităţile pentru două evenimente discrete şi complementare.

9.3.2 Algoritmul de antrenare

1. **Asamblarea setului de antrenare în X si y.** Fie *X* matricea cu trăsături pentru setul de antrenare. În acest caz *X* va conţine pe

fiecare linie valorile binarizate ale unei imagini din setul de antrenare. Se stochează 0 sau 255 în funcţie după cum pixelul pe poziţia respectivă este mai mic, respectiv mai mare sau egal decât un prag fixat la 128. *X* are dimensiunea de *n* x *d*, unde *n* este numărul de instanţe de antrenare, iar *d = 28* x *28* este mărimea unei imagini. Vectorul de etichete *y* are dimensiunea *n* x *1*.

2. **Calcularea vectorului a priori.** Probabilitatea *a priori* pentru clasa *i* se calculează ca şi fracţia instanţelor aparţinând clasei *i* dintre toate instanţele de antrenare:

$$P(c = i) = {}^{n_i}/_{n}$$

3. **Calcularea matricii de verosimilitate.** Funcţia de verosimilitate pentru ca trăsătura x_j să fie egală cu 255 condiţionată de clasa *c* este egală cu numărul de instanţe din clasa *c* care au trăsătura x_j egală cu 255 împărţit la numărul total de instanţe din clasa *c*:

$$p(x_j = 255 | c = i) = \frac{count(X_{kj} = 255 \wedge y_k = i)}{n_i}$$

Funcția de verosimilitate pentru ca trăsătura x_j să fie egală cu 0, se poate deduce utilizând valoarea anterioră, astfel:

$$p(x_j = 0 | c = i) = 1 - p(x_j = 255 | c = i)$$

Pentru a evita înmulţirea cu 0 la produsul final trebuie să ne asigurăm că verosimilitățile nu au valoarea 0. Putem rezolva această problemă prin modificarea la 10^{-5} a tuturor valorilor care sunt sub 10^{-5}. O alternativă practică este utilizarea netezirii Laplace, unde $|C|$ este numărul de clase:

$$p(x_j = 255 | c = i) = \frac{count(X_{kj} = 255 \wedge y_k = i) + 1}{n_i + |C|}$$

9.3.3 Algoritmul de clasificare

Odată ce s-au calculat valorile pentru probabilitățile *a priori* şi funcțiile de verosimilitate, se poate efectua clasificarea. Deoarece valorile probabilităţilor sunt cuprinse între 0 şi 1 produsul lor va fi un număr foarte mic. Pentru a evita problemele de precizie numerică se va lucra cu logaritmul probabilităţilor posterioare. Fie *T* vectorul de trăsături pentru imaginea de test, care sunt valorile binarizate în acelaşi mod ca şi la imaginile antrenare. Logaritmul probabilităţii posterioare se va calcula pentru fiecare clasă ca şi:

$$\log\big(p(C=i|T)\big) \propto \log\big(P(C=i)\big) + \sum_{j=1}^{d} \log\Big(p\big(x_j = T_j \big| C=i\big)\Big)$$

Următorul pas este alegerea clasei cu logaritmul probabilității maxime. Deoarece ordinea probabilităţilor posterioare nu se schimbă datorită logaritmului, clasificatorul va selecta clasa cu maximul logaritmului probabilității posterioare, la fel ca şi în teoria prezentată.

9.4 Detalii de implementare

Pentru a încărca primele 100 de imagini din clasa *c*:

```
char fname[256];
int c = 1;
int index = 0;
while(index < 100){
      sprintf(fname, "train/%d/%06d.png", c, index);
      Mat img = imread(fname, IMREAD_GRAYSCALE);
      if (img.cols==0) break;
      //procesarea imaginii curente
      index++;
}
```

Probabilitatea *a priori* este un vector de *C x 1*:

```
const int C = 3; //numărul de clase
Mat priors(C,1,CV_64FC1);
```

Verosimilitatea este o matrice de *C x d* (memorăm doar verosimilitatea pentru valoarea 255):

```
const int d = 28*28;
Mat likelihood(C,d,CV_64FC1);
```

Sugestie de antet pentru funcţia de clasificare:

```
int classifyBayes(Mat img, Mat priors, Mat likelihood);
```

9.5 Activitate practică

1. Încărcaţi imaginile din setul de antrenare şi aplicaţi operaţia de binarizare. Salvaţi valorile în matricea de trăsături. Salvaţi clasa instanţelor în vectorul de etichete *y*. Pentru varianta inițială utilizaţi doar primele 100 de imagini din clasele 0 şi 1.
2. Implementaţi algoritmul de antrenare.

2.1. Calculaţi şi salvaţi probabilitatea *a priori* pentru fiecare clasă.

2.2. Calculaţi şi salvaţi verosimilitatea pentru fiecare clasă şi fiecare trăsătură. Aplicaţi netezirea Laplace pentru a evita valorile nule.

3. Aplicați clasificatorul Bayes Naiv pe o imagine necunoscută.

4. La clasificare afişaţi logaritmul probabilităţilor posterioare pentru fiecare clasă. Opţional, convertiţi aceste valori în probabilităţi.

5. Evaluaţi clasificatorul pe setul de test. Calculaţi si afişaţi matricea de confuzie şi eroarea de clasificare. Eroarea este egală cu numărul de instanţe clasificate greşit per numărul total de instanţe.

6. Antrenaţi şi evaluaţi clasificatorul pe setul întreg de date, utilizând toate clasele.

9.6 Referinţe

[1] *Data Science Textbook* – https://docs.tibco.com/data-science/textbook

[2] LeCun, Yann, Corinna Cortes, and Christopher JC Burges. "The MNIST database." URL http://yann. lecun. com/exdb/mnist (1998).

10 Clasificatori liniari şi algoritmul perceptron

10.1 Obiective

În această lucrare de laborator se prezintă algoritmul de învăţare perceptron pentru clasificatori liniari. Se vor aplica metodele gradient descent şi stochastic gradient descent pentru a determina vectorul de ponderi. Se va rezolva o problemă de clasificare binară folosind două trăsături.

10.2 Fundamente teoretice

Scopul clasificării este determinarea clasei unor exemple descrise prin trăsături. Clasa unui exemplu face parte dintr-o mulţime finită, de exemplu: roşu/albastru, pieton/fundal, o cifră din mulţimea 0, 1, ... 9. Un clasificator liniar reuşeşte acest lucru prin luarea deciziei în funcţie de o combinaţie liniară a trăsăturilor.

10.2.1 Definiţii

Se defineşte un set de date etichetat ca şi o pereche (*X, Y*), unde $X \in M_{n\times m}(R)$ este o matrice cu *n* linii şi *m* coloane având valori reale şi Y este un vector coloană $Y \in M_{n\times 1}(D)$, care conţine etichetele pentru fiecare instanţă, iar D este mulţimea claselor posibile. X conţine pe linii instanţele de antrenare descrise cu *m* trăsături. Un clasificator este o funcţie care mapează orice instanţă reprezentată printr-un vector de trăsături la o clasă: $f: R^m \to D$.

Astfel un clasificator separă spaţiul de trăsături în $|D|$ regiuni disjunte (numărul de elemente din D). Subspaţiul care separă clasele se numeşte suprafaţa de decizie (eng. *decision boundary*). Dacă problema de clasificare este bidimensională acest subspaţiu va fi unidimensional şi va fi o linie sau o curbă, în general. În continuare vom discuta despre cazul cu două clase, algoritmul se poate extinde uşor la mai multe clase. Pentru etichete vom folosi $D = \{-1,1\}$.

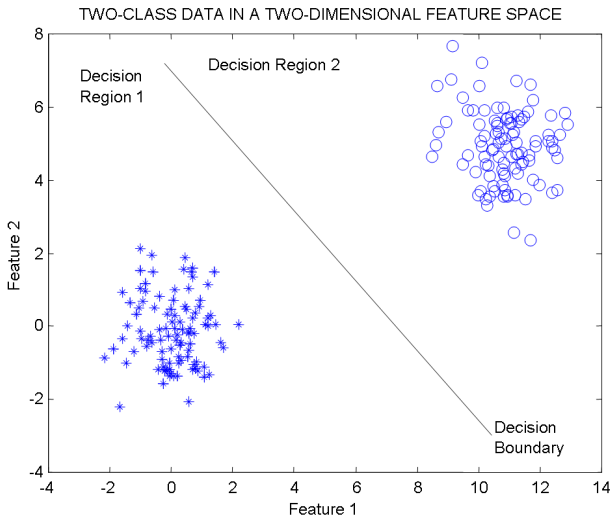


Figura 10.1. Exemplu de clasificator liniar care separă două clase şi utilizează două trăsături

10.2.2 Forma generală a unui clasificator liniar

Un clasificator liniar este o combinaţie liniară a trăsăturilor de intrare. Dacă notăm cu $\boldsymbol{x} \in M_{m\times 1}(R)$ vectorul de trăsături şi cu $\boldsymbol{w} \in M_{m\times 1}(R)$ vectorul de ponderi, putem exprima funcţia de clasificare ca şi:

$$g(\boldsymbol{x}) = w_0 + \sum_{i=1}^{m} w_i x_i = w_0 + \boldsymbol{w}^T \boldsymbol{x}$$

unde

- $\boldsymbol{w}$ este **vectorul de ponderi**
- w_0 este deplasamentul sau valoarea de prag

Următoarea schemă ilustrează modul de operare a clasificatorului liniar:

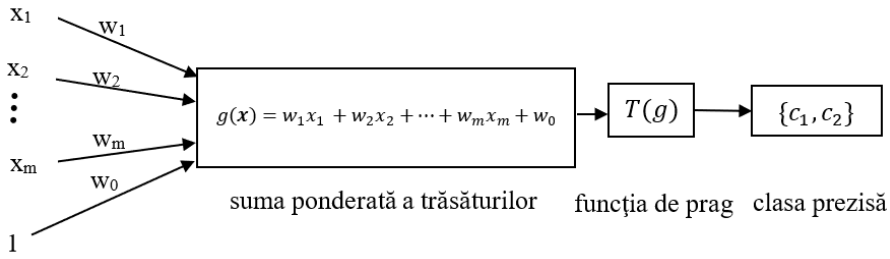


Figura 10.2 – Schema pentru un clasificator liniar

Pentru a simplifica notaţia, calculele şi pentru a permite tratarea generală vom absorbi valoarea w_0 în vectorul de ponderi şi vom augmenta vectorul de trăsături cu o componentă egală cu 1. În acest caz se simplifică expresia în:

$$w_0 + \boldsymbol{w}^T\boldsymbol{x} = [w_0 \quad \boldsymbol{w}]\begin{bmatrix}1\\ \boldsymbol{x}\end{bmatrix} = \bar{\boldsymbol{w}}^T\bar{\boldsymbol{x}}$$

Un clasificator liniar pentru două clase implementează următoarea regulă de clasificare:

- dacă $g(\boldsymbol{x}) > 0$, decidem că $\boldsymbol{x}$ aparţine clasei +1;
- dacă $g(\boldsymbol{x}) < 0$, decidem că $\boldsymbol{x}$ aparţine clasei -1;

sau echivalent:

- dacă $w^T\boldsymbol{x} > -w_0$, decidem că $\boldsymbol{x}$ aparţine clasei +1;
- dacă $w^T\boldsymbol{x} < -w_0$, decidem că $\boldsymbol{x}$ aparţine clasei -1;

Dacă $g(\boldsymbol{x}) = 0$, atunci $\boldsymbol{x}$ poate fi atribuit la oricare clasă, fiind pe suprafaţa de separare.

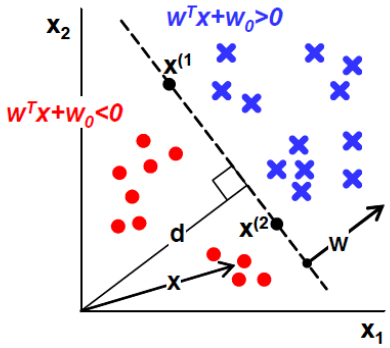


Figura 10.3 Ilustrarea componentelor din vectorul de ponderi: vectorul $\boldsymbol{w}$ este normala la dreapta de separare iar $\boldsymbol{w_0}$ este egal cu distanţa d (cu semn) a dreptei de la origine înmulţită cu $\|\boldsymbol{w}\|$ norma lui $\boldsymbol{w}$

10.2.3 Metode de învăţare pentru clasificatori liniari

Vom prezenta două metode de învăţare pentru clasificatori liniari. Învăţarea se face prin transformarea problemei de clasificare într-o problemă de minimizare a unei funcţii de cost. Denumim funcţia de cost L (eng. *loss function*). Funcţia poate să aibă mai multe forme şi

trebuie să penalizeze diferenţele dintre predicţia clasificatorului şi clasa adevărată. În cazul algoritmului perceptron adoptăm următoarea funcţie de cost:

$$L(\overline{\boldsymbol{w}}) = \frac{1}{n}\sum_{i=1}^{n} max(0, -y_i \overline{\boldsymbol{w}}^T \cdot \overline{\boldsymbol{x}}_i) = \frac{1}{n}\sum_{i=1}^{n} L_i\,(\overline{\boldsymbol{w}})$$

Dacă o instanţă este corect clasificată atunci nu se aplică nici o penalizare, în schimb, dacă instanţa se atribuie la clasa greşită atunci se penalizează proporţional cu scorul greşit $y_i \overline{\boldsymbol{w}}^T \cdot \overline{\boldsymbol{x}}_i$. Semnul expresiei $y_i \overline{\boldsymbol{w}}^T \cdot \overline{\boldsymbol{x}}_i$ este negativ dacă semnul etichetei de clasă y_i este diferit de semnul clasei prezise $g(\boldsymbol{x})$.

Pentru a minimiza acest cost vom utiliza metoda gradient descent. Vom porni de la un vector de ponderi aleatoriu şi vom schimba acest vector bazându-ne pe valoarea gradientului în poziţia curentă. Pasul va fi în direcţia opusă a gradientului:

$$\overline{\boldsymbol{w}}_{k+1} \leftarrow \overline{\boldsymbol{w}}_k - \eta \boldsymbol{\nabla L}(\overline{\boldsymbol{w}}_k)$$

unde $\overline{\boldsymbol{w}}_k$ este vectorul de ponderi la momentul *k*, parametrul η controlează mărimea pasului şi se numeşte rata de învăţare (eng. *learning rate*), şi $\boldsymbol{\nabla L}(\overline{\boldsymbol{w}}_k)$ este vectorul de gradient calculat în punctul $\overline{\boldsymbol{w}}_k$. Vectorul de gradient are expresia:

$$\boldsymbol{\nabla L}(\overline{\boldsymbol{w}}) = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{\nabla L_i}(\overline{\boldsymbol{w}})$$

$$\boldsymbol{\nabla L_i}(\overline{\boldsymbol{w}}) = \begin{cases} 0, & \text{daca } y_i \overline{\boldsymbol{w}}^T \cdot \overline{\boldsymbol{x}}_i > 0 \\ -y_i \overline{\boldsymbol{x}}_i\ , & \text{altfel} \end{cases}$$

În metoda care foloseşte gradientul pe setul întreg de date, ponderile se schimbă doar după ce s-au vizitat toate exemplele de antrenare. Această abordare se mai numeşte şi *batch-update*. Algoritmul perceptron din lucrarea [1] actualizează ponderile după examinarea fiecărui exemplu de antrenare. În acest caz algoritmul se numeşte *online perceptron*. Regula de actualizare devine:

$$\overline{\boldsymbol{w}}_{k+1} \leftarrow \overline{\boldsymbol{w}}_k - \eta \boldsymbol{\nabla L_i}(\overline{\boldsymbol{w}})$$

Redăm în continuare, în paralel, cele două variante ale algoritmului de învăţare:

Algoritmul Batch Perceptron	**Algoritmul Online Perceptron**

init **w**, η, E_{limit}, max_iter
for iter=1:max_iter
  E = 0, L = 0
  $\nabla \boldsymbol{L}$ = [0,0,0]
  for i=1:n
  $z_i = \sum_{j=0}^{d} w_j X_{ij}$
  if $z_i \cdot y_i \leq 0$
  $\nabla \boldsymbol{L} \leftarrow \nabla \boldsymbol{L} - y_i \boldsymbol{X}_i$
  $E \leftarrow E + 1$
  $L \leftarrow L - y_i z_i$
  endif
  endfor
  $E \leftarrow E/n$
  $L \leftarrow L/n$
  $\nabla \boldsymbol{L} \leftarrow \nabla \boldsymbol{L}/n$
  if $E < E_{limit}$
  break
  $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \nabla \boldsymbol{L}$
endfor
	init **w**, η, E_{limit}, max_iter
	for iter=1:max_iter
	  E = 0
	  for i=1:n
	  $z_i = \sum_{j=0}^{d} w_j X_{ij}$
	  if $z_i \cdot y_i \leq 0$
	  $\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta \boldsymbol{X}_i y_i$
	  $E \leftarrow E + 1$
	  endif
	  endfor
	  $E \leftarrow E/n$
	  if $E < E_{limit}$
	  break
	endfor

10.3 Aspecte practice

Vom folosi puncte 2D care sunt împărţite în 2 clase. Punctele se citesc din imaginile de intrare şi apartenenţa lor se stabileşte pe baza culorii. Asociem clasa -1 la punctele albastre şi 1 la punctele roşii.

Fiecare punct va fi descris de două trăsături x_1 şi x_2 care coincid cu coordonatele lor *x* şi respectiv *y*. Astfel, vectorul de trăsături augmentat are forma $\boldsymbol{x} = [1\ x_1\ x_2]^T$ iar vectorul de ponderi are trei componente $\overline{\boldsymbol{w}} = [w_0\ w_1\ w_2]^T$.

10.4 Activitate practică

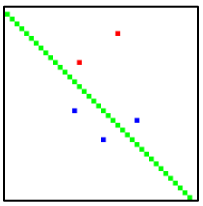
1. Citiţi punctele dintr-o singură imagine *test0*.bmp* şi construiţi setul de antrenare (*X, Y*). Asociaţi -1 la punctele albastre şi +1 la punctele roşii.

2. Implementaţi algoritmul *Online Perceptron* pentru a găsi linia de separare dintre cele două clase. Sugestie pentru parametri: $\eta=10^{-4}$, $w_0 = [0.1, 0.1, 0.1]^T$, $E_{limit}=10^{-5}$, $max\_iter = 10^5$.
   Observaţie: Pentru a accelera convergența algoritmului se folosește o rată de învățare mai mare doar pentru termenul liber w_0.
3. Desenaţi linia de separare găsită de algoritm, dată de ecuaţia:
$$w_0 + w_1 x + w_2 y = 0$$
4. Implementaţi algoritmul *Batch Perceptron*. Găsiţi parametri buni care asigură învăţarea. Se va monitoriza funcţia de cost care trebuie să descrească încet la fiecare pas.
5. Vizualizaţi linia de separare în timp ce rulează algoritmul pentru a observa modul în care aceasta se mută.
6. Schimbaţi valorile de start pentru vectorul $\boldsymbol{w}$, modificaţi rata de învăţare şi observaţi ce se întâmplă în fiecare caz. Ce înseamnă dacă funcţia de cost L oscilează în loc să descrească la fiecare pas?

10.5 Exemplu numeric

Fie punctele din fişierul *test00.bmp* – (*x*, *y*) sau (*coloana*, *rând*):

- Roșii: (23, 5), (15,11) – clasa +1
- Albastre: (14, 21), (27,23), (20, 27) – clasa -1



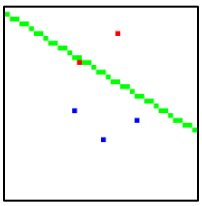
Pașii pentru algoritmul online perceptron sunt următorii:

- Rata de învățare = 0.01

Iterația 0

i=0: w=[1.000000 1.000000 -1.000000] xi=[1 23 5] yi = 1 zi=19.000000
i=1: w=[1.000000 1.000000 -1.000000] xi=[1 15 11] yi = 1 zi=5.000000
i=2: w=[1.000000 1.000000 -1.000000] xi=[1 14 21] yi = -1 zi=-6.000000
i=3: w=[1.000000 1.000000 -1.000000] xi=[1 27 23] yi = -1 zi=5.000000 greșit
  - update w0 = w0 - 0.01, w1 = w1 - 27*0.01, w2 = w2 – 23*0.01

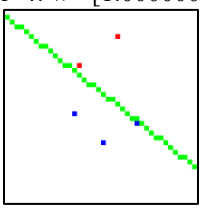
i=4: w=[0.990000 0.730000 -1.230000] xi=[1 20 27] yi = -1 zi=-17.620000



Iterația 1

i=0: w= [0.990000 0.730000 -1.230000] xi= [1 23 5] yi = 1 zi=11.630000
i=1: w= [0.990000 0.730000 -1.230000] xi= [1 15 11] yi = 1 zi=-1.590000 greșit
- update w0 = w0 + 0.01, w1 = w1 + 15*0.01, w2 = w2 + 11*0.01

i=2: w= [1.000000 0.880000 -1.120000] xi= [1 14 21] yi = -1 zi=-10.200000
i=3: w= [1.000000 0.880000 -1.120000] xi= [1 27 23] yi = -1 zi=-1.000000
i=4: w= [1.000000 0.880000 -1.120000] xi= [1 20 27] yi = -1 zi=-11.640000



Iterația 2

i=0: w= [1.000000 0.880000 -1.120000] xi= [1 23 5] yi = 1 zi=15.640000
i=1: w= [1.000000 0.880000 -1.120000] xi= [1 15 11] yi = 1 zi=1.880000
i=2: w= [1.000000 0.880000 -1.120000] xi= [1 14 21] yi = -1 zi=-10.200000
i=3: w= [1.000000 0.880000 -1.120000] xi= [1 27 23] yi = -1 zi=-1.000000
i=4: w= [1.000000 0.880000 -1.120000] xi= [1 20 27] yi = -1 zi=-11.640000
Toate clasificate corect

10.6 Referinţe

[1] Rosenblatt, Frank (1957), The Perceptron - a perceiving and recognizing automaton. Report 85-460-1, Cornell Aeronautical Laboratory.

[2] Richard O. Duda, Peter E. Hart, David G. Stork: Pattern Classification 2nd ed.

[3] Xiaoli Z. Fern, Machine Learning Course, Oregon University http://web.engr.oregonstate.edu/~xfern/classes/cs434/slides/perceptron-2.pdf

[4] Gradient Descent - http://en.wikipedia.org/wiki/Gradient_descent

[5] Avrim Blum, Machine Learning Theory, Carnegie Mellon University - https://www.cs.cmu.edu/~avrim/ML10/lect0125.pdf

11 Metoda AdaBoost

11.1 Obiective

În acestă lucrare de laborator se va studia un clasificator compus folosind metoda AdaBoost (*Adaptive Boosting*). Se va aplica acest clasificator pe o problemă de clasificare binară a unei mulţimi de puncte 2D.

11.2 Fundamente teoretice

AdaBoost, abreviere pentru *Adaptive Boosting*, este o meta-metodă de învăţare formulată de Yoav Freund şi Robert Schapire în [1], care au câştigat premiul Gödel din 2003 pentru activitatea lor [2]. În acest laborator vom separa puncte 2D în două clase diferite, clasa punctelor fiind dată de culoarea lor.

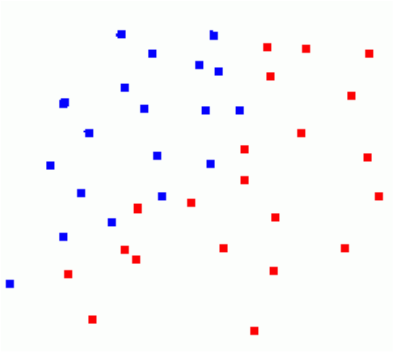


Figura 11.1 Exemplu de puncte din două clase

Ideea generală a algoritmului AdaBoost este construirea unui clasificator puternic $H_T(\boldsymbol{x})$ care este o combinaţie liniară a T clasificatori slabi denumiţi h_t:

$$H_T(\boldsymbol{x}) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x})\right)$$

Fiecare clasificator slab returnează +1 sau -1 şi este ponderat cu α_t. Rezultatul final este dat de semnul sumei. Se va utiliza un arbore de decizie cu un singur nivel (*decision stump*) ca și clasificator slab. Acesta clasifică o instanţă pe baza unei singure trăsături. Dacă valoarea

trăsăturii este sub un prag se returnează -1 iar dacă este peste, se returnează +1.

Se dă setul de antrenare sub forma unei matrice de trăsături $\boldsymbol{X}$ de dimensiune n x m, care conţine n exemple de antrenare, fiecare rând reprezentând un exemplu individual de dimensiune m. În cazul nostru, m este 2 şi vom folosi ca trăsături coordonatele punctelor. Vectorul de etichete y conţine clasele punctelor. Vom considera punctele roşii ca având clasa +1 şi punctele albastre clasa -1.

Metoda asociază o pondere fiecărui exemplu de antrenare. Vom stoca aceste ponderi în vectorul $\boldsymbol{w}$ de dimensiune n. Iniţial toate instanţele au aceeaşi pondere de $1/n$. Descriem în continuare algoritmul AdaBoost pentru a obţine clasificatorul puternic H_T.

Algorithm AdaBoost

```
init wᵢ=1/n
h = []
α = []
for t=1:T  //T is the number of weak learners (wl)
  h_t = findWeakLearner(X,y,w)
  α_t = 0.5 ln((1−ε_t)/ε_t)  //ε_t is the error of the wl h_t
  s = 0 //s is the sum of the weights
  for i=1:n //n is the nr. of samples in the dataset
    //wrongly classified examples
    //will have y_i h_t(X_i) < 0
    //their weights will be higher in the next step
    w_i ← w_i · exp(−α_t y_i h_t(X_i))  //h_t classifies row X_i
    s += w_i
  endfor
  for i=1:n
    w_i ← w_i/s  //normalize the weights
  endfor
endfor
//returns all the alpha values and the weak learners
return [α,h]
```

```
findWeakLearner(X,y,w)
best_h = {}
best_err = ∞
for j=1:X.cols
//img.size is img rows for j=0 and img cols for j=1
  for threshold=0:img.size
    for class_label={-1,1}
      e=0
      for i=1:X.rows
        if X(i,j)<threshold
          z_i=class_label
        else
          z_i=-class_label
        endif
        if z_i y_i < 0
          e += w_i
        endif
      endfor
      if e<best_err
        best_err = e
        best_h = {j,threshold,class_label,e}
      endif
    endfor
  endfor
endfor
return best_h
```

Ideea care stă la baza algoritmului este de a găsi cel mai bun clasificator slab pe baza erorii ponderate (cea mai mică valoare) şi apoi să modificăm importanţa exemplelor. Exemplele clasificate greşit vor avea o pondere mai mare, deci vor conta mai mult pentru selecția următorilor clasificatori slabi, iar cele corecte o pondere mai mică. Un exemplu de antrenare e clasificat greșit dacă expresia $y_i h_t(X_i)$ este negativă (eticheta de clasă şi predicţia noastră au semne diferite).

La pasul următor vom găsi un alt clasificator slab fiindcă acesta trebuie să clasifice corect instanţele cu pondere mai mare deorece acestea au o contribuţie mai mare în eroarea ponderată. Fiecare clasificator slab influenţează scorul final. Această influenţă este ponderată în funcţie de performanţa lui pe setul de antrenare.

11.3 Detalii de implementare

Structură sugerată pentru un clasificator slab:

```
struct weaklearner{
  int feature_idx;
  int threshold;
  int class_label;
  float error;

  int classify(Mat X_row){
    if (X_row.at<float>(feature_idx) < threshold)
      return class label;
    else
      return -class_label;
  }
};
```

Antetul funcţiei care returnează cel mai bun clasificator slab:

```
weaklearner findWeakLearner(Mat X, Mat y, Mat w)
```

Structură sugerată pentru clasificatorul puternic (MAXT este o constantă):

```
struct classifier{
  int T;
  float alphas[MAXT];
  weaklearner hs[MAXT];

  int classify(Mat X_row){
    // TODO implement classification
  }
};
```

Antetul funcţiei care colorează fundalul (să nu modificaţi imaginea originală):

```
void drawBoundary(Mat img, classifier clf)
```

11.4 Activitate practică

1. Citiţi datele de antrenare dintr-o singură imagine (*points*.bmp*). Fiecare rând din matricea ***X*** trebuie să conţină rândul şi coloana unui punct colorat din imagine. Clasa punctului se memorează în ***y*** considerând +1 pentru roşu şi -1 pentru albastru.
2. Implementaţi clasificatorul slab.
3. Implementaţi funcţia `findWeakLearner`.
4. Implementaţi funcţia `drawBoundary` care colorează fundalul în funcţie de apartenenţă. Fiecare pixel alb se colorează cu galben dacă face parte din clasa +1 şi cu turcoaz pentru -1. Testaţi funcţia cu un clasificator puternic format dintr-un singur clasificator slab.
5. Implementaţi algoritmul AdaBoost pentru a găsi un clasificator puternic format din *T* clasificatori slabi. Vizualizaţi bordura de decizie. Găsiţi parametrul *T* pentru care obtineţi eroare 0. Care sunt limitările metodei?

11.5 Exemple de rezultate

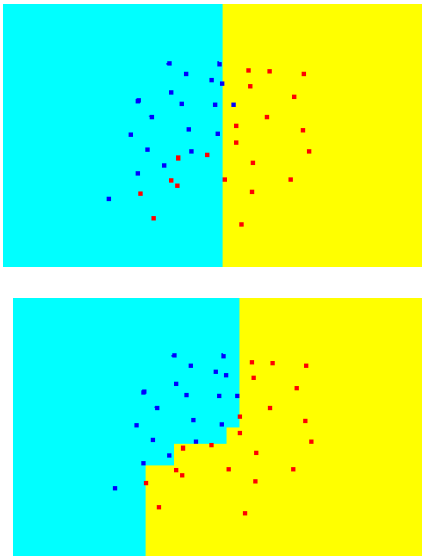


Figura 10.2. Rezultate obţinute pe points1 cu T=1 (stânga) şi T=13 (dreapta)

11.6 Referinţe

[1] Robert E. Schapire, The Boosting Approach to Machine Learning, An Overview, 2001

[2] AdaBoost - https://en.wikipedia.org/wiki/AdaBoost

12 Clasificare cu Support Vector Machine

12.1 Obiective

În această lucrare se va implementa clasificatorul Support Vector Machine (SVM) liniar și se vor studia mecanismele de clasificare bazate pe vectori suport și clasificatori cu soft margin.

12.2 Fundamente teoretice

12.2.1. Clasificatori cu separare strictă (hard-margin)

Pentru a explica ce înseamnă un clasificator SVM cu separare strictă se va porni de la un exemplu simplu de problema de clasificare. Se dorește separarea liniară a unei mulțimi de puncte în spațiul cartezian bidimensional. Un exemplu este prezentat în Figura 12.1.

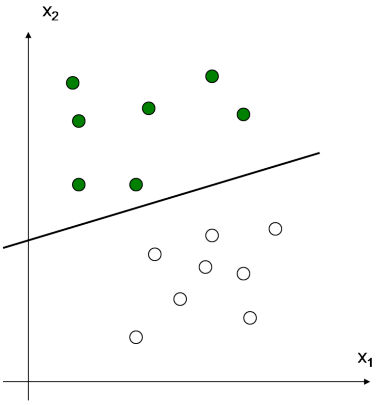


Figura 12.1. O mulțime de puncte din două clase liniar separabile

Cum putem clasifica aceste puncte folosind o funcție discriminantă liniară care minimizează eroarea de clasificare? Există un număr infinit de răspunsuri, câteva drepte separatoare cu eroare 0 sunt ilustrate in Figura 12.2.

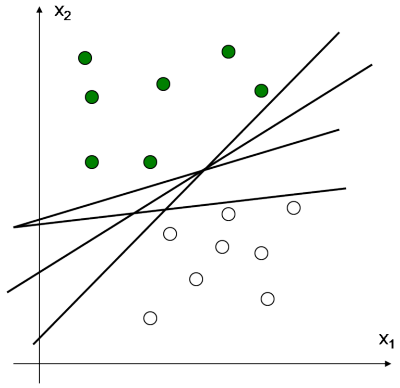


Figura 12.2. Exemplu de clasificatori liniari cu eroare de clasificare zero

Din această mulțime de soluții este necesar să determinăm clasificatorul optim. Pentru SVM considerăm clasificatorul optim clasificatorul care maximizează zona de margine. Zona de margine este definită ca și lățimea bandei la care poate fi extinsă linia de separare înainte să întâlnim un punct de date. (vezi Figura 12.3).

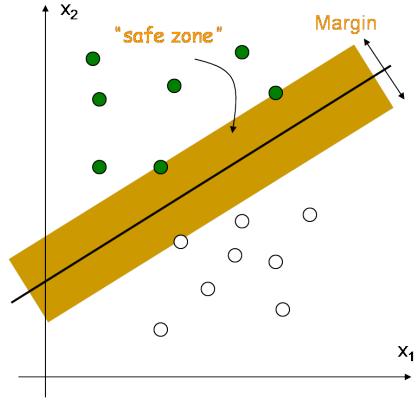


Figura 12.3. Zona de margine pentru un clasificator liniar

Clasificatorul cu zona de margine maximală este considerat optim fiindcă este robust la zgomot și are o abilitate de generalizare puternică.

Fiind date o mulțime de puncte etichetate: $\{x_i, y_i\}, i = 1,2, \dots, n$ unde

$$y_i = +1, w^T x_i + b > 0$$
$$y_i = -1, w^T x_i + b < 0$$

Folosind o transformare de scală aplicată pe w si b, condițiile sunt echivalente cu:

$$y_i = +1, w^T x_i + b \geq 1$$
$$y_i = -1, w^T x_i + b \leq -1$$

Se știe că:

$$w^T x^+ + b = 1$$
$$w^T x^- + b = -1$$

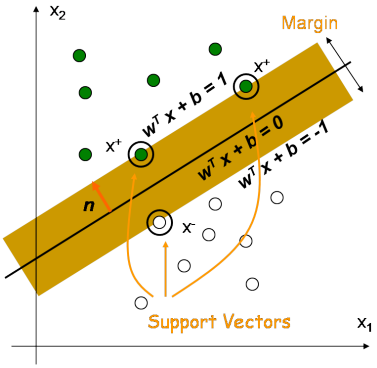


Figura 12.4. Exemplele pozitive și negative cele mai apropiate de zona de margine sunt vectorii suport

Lățimea zonei de separare este:

$$M = (x^+ - x^-) \cdot n = (x^+ - x^-) \cdot \frac{w}{||w||} = \frac{2}{||w||}$$

Problema maximizării acestei expresii este dificilă fiindcă depinde de norma vectorului w, ||**w**||, care conține radical. Se poate minimiza $\frac{1}{2}\|w\|^2$ în locul normei, care duce la o problemă de optimizare echivalentă dar care se poate rezolva.

Cu această schimbare rezultă o problemă de optimizare pătratică: Să se minimizeze expresia $\frac{1}{2}\|w\|^2$ cu constrângerile:

- $y_i = +1, w^T x_i + b \geq 1$
- $y_i = -1, w^T x_i + b \leq -1$

O reformulare mai succintă este: Să se minimizeze $\frac{1}{2}\|w\|^2$ cu constrângerile $y_i(w^T x_i + b) \geq 1$. Soluția la această problemă de optimizare se găsește în metoda multiplicatorilor lui Lagrange.

12.2.2 Clasificatori cu separare permisivă (soft-margin)

In 1995, Corinna Cortes si Vladimir Vapnik au propus un alt criteriu mai permisiv care permite exemple etichetate greșit. Daca nu există un hiperplan care separă exemplele în două clase atunci un clasificator cu separare permisivă va alege hiperplanul care separă datele cât mai corect, maximizând distanța de la cel mai apropiat exemplu corect clasificat. Această metodă introduce variabilele auxiliare, ξ_i, care măsoară gradul de clasificare greșită a punctului x_i.

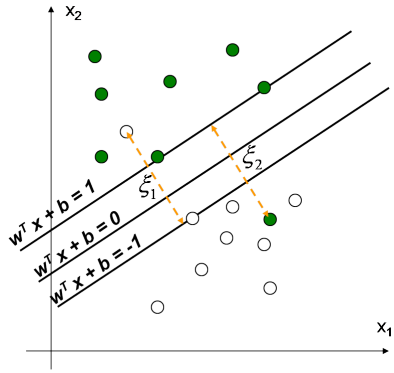


Figura 12.5 Clasificatoare cu separare permisivă

Prin minimizarea sumei $\sum_i \xi_i$, obținem ξ_i ca fiind:

$$\begin{cases} \mathbf{w}^T\mathbf{x}_i + b \geq 1 - \xi_i & y_i = 1 \\ \mathbf{w}^T\mathbf{x}_i + b \leq -1 + \xi_i & y_i = -1 \\ \xi_i \geq 0 & \forall i \end{cases}$$

- ξ_i sunt variabilele auxiliare de optimizare;
- ξ_i=0 daca $\mathbf{x}_i$ este clasificat corect, și
- C este numărul maxim de erori

Trebuie sa minimizam $\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i$, supus la constrângerile $y_i(w^T x_i + b) \geq 1 - \xi_i$ si $\xi_i \geq 0$. Parametrul C poate fi văzut ca și un parametru care reglează eroarea față de zona de separare.

12.2.3 Support Vector Machine cu mapare neliniară (kernel trick)

În cazul în care datele nu sunt liniar separabile se poate aplica o transformare $x_i \rightarrow \phi(x_i)$ care mapează fiecare punct într-un spațiu cu dimensionalitate mai mare. Se dorește ca în acest spațiu punctele sa devină liniar separabile.

Fiind date o mulțime de perechi instanță și eticheta (x_i, y_i); i = 1… l unde $x_i \in R^n$ și y∈{+1,-1}[1], clasificatorul SVM rezolvă următoarea problemă de optimizare:

$$\text{minimizarea } \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n}\xi_i \text{ astfel încât}$$
$$y_i(\boldsymbol{w}^T\phi(\boldsymbol{x}_i) + b) \geq 1 - \xi_i \text{ and } \xi_i \geq 0.$$

În acest caz vectorii de antrenare x_i sunt mapați într-un spațiu cu dimensionalitate mai mare (posibil infinită). Se găsește hiperplanul separator optimal în acest spațiu. $C > 0$ este termenul care penalizează erorile de clasificare.

Se poate reformula problema de optimizare astfel încât să folosim o funcție kernel, în loc să mapăm vectorii. Se definește funcția de kernel (nucleu):

$$K(x_i, x_j) = \Phi(x_i)^T \Phi(x_j)$$

In literatură există o multitudine de funcții kernel, se recomandă folosirea următoarelor tipuri: liniar, polinomial, radial basis function și sigmoid.

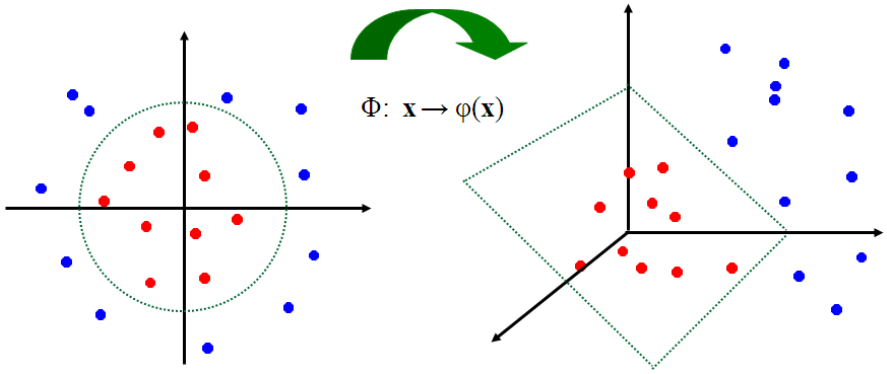


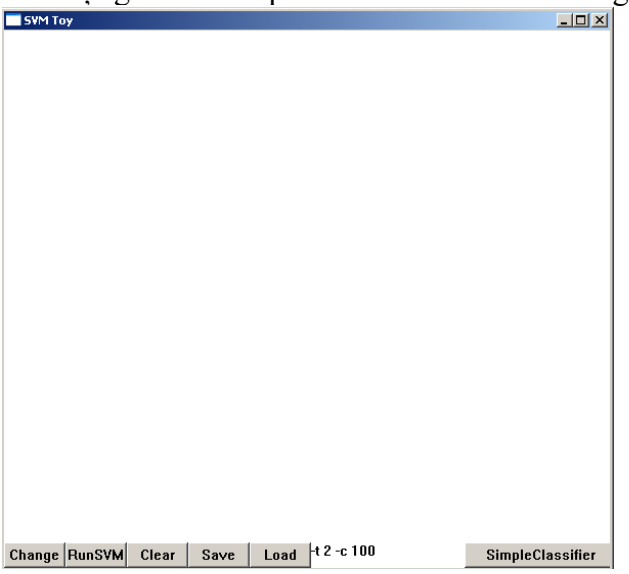
Figura 12.6 Support Vector Machine cu mapare neliniară

12.3 Activitate practică

Ca și exercițiu veți folosi aplicația denumită SVM-toy de la LIBSVM tools [3], care este o implementare in C++ a clasificatorilor cu separare permisivă folosind diferite funcții de kernel.

1. Se descarcă fișierul *zip* care conține proiectul și se încarcă în Visual Studio.

2. Interfața grafică este prezentată în următoarea figură:



Butoanele de la interfață au următoarele funcționalități:

- 'Change': schimbă clasa punctului care va fi adăugat în spațiul de clasificare. Se poate adăuga un punct nou prin click stânga în zona ferestrei albe. Numărul maxim de clase este 3 care corespunde la 3 culori diferite
- 'RunSVM': rulează antrenarea unui SVM cu parametri specificați în câmpul editabil de lângă butonul 'Load'
- 'Clear': șterge toate punctele din spațiul de clasificare
- 'Save': salvează punctele din spațiul de clasificare curent într-un fișier
- 'Load': încarcă o imagine *bmp* și desenează punctele din spațiul de clasificare
- In câmpul editabil se specifică parametri clasificatorului, conține: '–t 2 –c 100'
  Vom folosi doi parametri:
  - '-t kernel_type' specifică tipul funcției kernel, are valoarea implicită de 2. Se poate alege dintre următoarele:
    0 – kernel liniar: u*v
    1 – kernel polinomial: (gamma*u'*v + coef0)^degree
    2 – radial basis function: exp(-gamma*|u-v|^2)
    3 – sigmoid: tanh(gamma*u'*v + coef0)
  - '-c cost' specifica valoarea pentru parametrul C

- ‘SimpleClassifier’ butonul lansează codul care urmează să fie implementat (se schimba *svm-toy.cpp*).

3. Pentru fiecare imagine din arhiva *zip* se rulează clasificatorul SVM cu parametri diferiți și se observă rezultatul.

4. Să se implementeze ‘SimpleClassifier’ și să se compare rezultatele cu un SVM cu kernel liniar.

   Codul se introduce în ramura instrucțiunii case din *svm-toy.cpp*:

```
case ID_BUTTON_SIMPLE_CLASSIFIER:
{
/* ************************************
      TO DO:
      WRITE   YOUR   CODE   HERE   FOR   THE   SIMPLE
CLASSIFIER
************************************ */
}
```

   Se menționează că punctele pentru clasificare sunt stocate într-o listă de puncte:

```
list<point> point_list;
```

   cu structura unui punct fiind definită:

```
struct point {
      double x, y;
      signed char value;
};
```

   Câmpul *value* corespunde clasei punctului (are valoarea 1, 2 sau 3).

   Coordonatele punctelor sunt normalizate între 0 și 1, iar punctul de origine (0,0) este localizat în colțul stânga sus.

   Dimensiunea spațiului de clasificare este de XLEN * YLEN, în consecință coordonatele reale corespunzătoare unui punct (x, y) vor fi (x*XLEN, y*YLEN).

Să se determine *w* si *b* folosind formulele următoare:

$$\mathbf{c}^+ = \frac{1}{m^+} \sum_{\{i:y_i=+1\}} \mathbf{x}_i$$

$$\mathbf{c}^- = \frac{1}{m^-} \sum_{\{i:y_i=-1\}} \mathbf{x}_i$$

$$\mathbf{c} = \frac{1}{2}(c^+ + c^-)$$

$$\mathbf{w} = c^+ - c^-$$

$$b = \langle c, c^+ \rangle - \langle c, c^- \rangle$$

și să se deseneze linia cu ecuația:

$$\langle w, \boldsymbol{x} \rangle - b = 0, \boldsymbol{x} \in R^2$$

Desenarea unei linii între două puncte se face cu metoda:

```
DrawLine(buffer_dc,x1, y1, x2, y2, RGB(255,0,0));
```

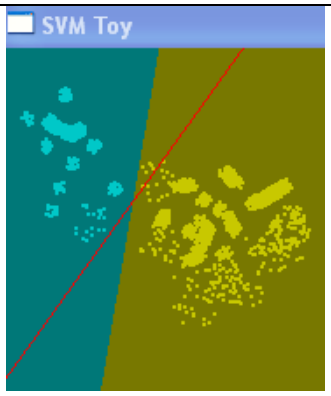
Se parcurge lista de puncte și se numără instanțele din clasa 1 și clasa 2 folosind următoarea secțiune de cod:

```
//declare an iterator
list<point>::iterator p;
int nrSamples1=0;
int nrSamples2=0;
double xC1=0,xC2=0,yC1=0,yC2=0;

for(p = point_list.begin(); p != point_list.end();
p++)
{
      if ((*p).value==1) //point from class '1'
      {
            nrSamples1++;
            xC1 =(*p).x; //normalized x coordinate
            yC1 =(*p).y; //normalized y coordinate
      }

      if ((*p).value==2)  //point from class '2'
      {
            nrSamples2++;
            xC2 =(*p).x; //normalized x coordinate
            yC2 =(*p).y; //normalized y coordinate
      }
}
```

12.3 Exemplu de rezultat



Detalii:

- punctele 2D care vor fi clasificate
- 2 clase, 2 trăsături (x_1 si x_2)
- separatorul generat de algoritmul 'Simple Classifier' este linia roșie
- separarea indicată prin culori diferite este cea obținută de un SVM cu kernel liniar (-t 0) si cost C=100 (-c 100)

Se observă:

- marginea maximizată obținută cu SVM
- punctele clasificate greșit cu clasificatorul simplu

12.4 Referințe

[1] J. Shawe-Taylor, N. Cristianini: *Kernel Methods for Pattern Analysis.* Pattern Analysis (Chapter 1)
[2] B. Scholkopf, A. Smola: *Learning with Kernels*. A Tutorial Introduction (Chapter 1), MIT University Press.
[3] LIBSVM: http://www.csie.ntu.edu.tw/~cjlin/libsvm/

REFERINTE

Referințe globale:

1. Richard O. Duda, Peter E. Hart , David G . Stork, "Pattern Clasification", John Wiley and Sons, 2001.

2. Kevin P. Murphy, “Machine Learning: A Probabilistic Perspective”, The MIT Press, 2012

3. Kevin P. Murphy, “Probabilistic Machine Learning: An Introduction”, The MIT Press, 2022

4. C.M. Bishop, “Pattern Recognition and Machine Learning”, second edition, Springer, 2016

5. S. Nedevschi, “Pattern Recognition Systems – Lecture Notes”, TUCN 2022

Referințe pe lucrări de laborator:

1.1 Stanford Machine Learning CS229 - lecture notes 1 – https://see.stanford.edu/materials/aimlcs229/cs229-notes1.pdf

1.2 Tomas Svoboda - Least-squares solution of Homogeneous Equation http://cmp.felk.cvut.cz/cmp/courses/XE33PVR/WS20072008/Lectures/Supporting/constrained_lsq.pdf

2.1 Robert C. Bolles, Martin A. Fischler: *A RANSAC-Based Approach to Model Fitting and Its Application to Finding Cylinders în Range Data*, 1981

2.2 Richard Hartley, Andrew Zisserman: *Multiple View Geometry în Computer Vision,* 2003

3.1 P. Hough, “Method and means for recognizing complex patterns”, US patent 3,069,654, 1962.

3.2 R. O. Duda and P. E. Hart, "Use of the Hough Transformation to Detect Lines and Curves in Pictures," *Comm. ACM*, Vol. 15, pp. 11–15, 1972.

3.3 D. H. Ballard, "Generalizing the Hough Transform to Detect Arbitrary Shapes", *Pattern Recognition*, Vol.13, No.2, p.111-122, 1981.

3.4 https://en.wikipedia.org/wiki/Hough_transform

4.1 Wikipedia The Free Encyclopedia – *Distance Transform*, http://en.wikipedia.org/wiki/Distance_transform

4.2 Compendium of Computer Vision – *Distance Transform*, http://homepages.inf.ed.ac.uk/rbf/HIPR2/distance.htm

5.1 MIT CBCL FACE dataset, http://www.ai.mit.edu/courses/6.899/lectures/faces.tar.gz

6.1 Wikipedia article PCA - https://en.wikipedia.org/wiki/Principal_component_analysis

6.2 Stanford CS229 Machine Learning course notes - https://cs229.stanford.edu/main_notes.pdf

6.3 Lindsay Smith - PCA tutorial - http://faculty.iiit.ac.in/~mkrishna/PrincipalComponents.pdf

6.4 PCA in R (animation of projection) - https://poissonisfish.wordpress.com/2017/01/23/principal-component-analysis-in-r/

7.1 Cluster analysis Wikipedia article - https://en.wikipedia.org/wiki/Cluster_analysis

7.2 K-means Wikipedia article - https://en.wikipedia.org/wiki/K-means_clustering

7.3 Arthur, David, and Sergei Vassilvitskii. "k-means++: The advantages of careful seeding." Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms. Society for Industrial and Applied Mathematics, 2007.

7.4 P. Arbelaez, M. Maire, C. Fowlkes and J. Malik. „Contour Detection and Hierarchical Image Segmentation", IEEE TPAMI, Vol. 33, No. 5, pp. 898-916, May 2011.

7.5 Image segmentation dataset: https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/resources.html

8.1 Wikipedia article - k-NN classifier https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

8.2 Andrew Ng - Machine Learning: Nonparametric methods & Instance-based learning http://www.cs.cmu.edu/~epxing/Class/10701-08s/Lecture/lecture2.pdf

9.1 *Data Science Textbook* – http://docs.tibco.com/data-science/textbook

9.2 LeCun, Yann, Corinna Cortes, and Christopher JC Burges. "The MNIST database." - http://yann.lecun.com/exdb/mnist

.

10.1 Rosenblatt, Frank (1957), The Perceptron - a perceiving and recognizing automaton. Report 85-460-1, Cornell Aeronautical Laboratory.

10.2 Richard O. Duda, Peter E. Hart, David G. Stork: Pattern Classification 2nd ed.

10.3 Xiaoli Z. Fern, Machine Learning Course, Oregon University – http://web.engr.oregonstate.edu/~xfern/classes/cs434/slides/perceptron-2.pdf

10.4 Gradient Descent - http://en.wikipedia.org/wiki/Gradient_descent

10.5 Avrim Blum, Machine Learning Theory, Carnegie Mellon University - https://www.cs.cmu.edu/~avrim/ML10/lect0125.pdf

11.1 Robert E. Schapire, The Boosting Approach to Machine Learning, An Overview, 2001

11.2 AdaBoost - https://en.wikipedia.org/wiki/AdaBoost

12.1 J. Shawe-Taylor, N. Cristianini: *Kernel Methods for Pattern Analysis.* Pattern Analysis (Chapter 1)

12.2 B. Scholkopf, A. Smola: *Learning with Kernels*. A Tutorial Introduction (Chapter 1), MIT University Press.

12.3 LIBSVM: http://www.csie.ntu.edu.tw/~cjlin/libsvm/