Sisteme de recunoaştere a formelor – Lab 10

Metoda AdaBoost

1. Obiective

În acest laborator vom studia un clasificator compus folosind metoda AdaBoost (Adaptive Boosting). Vom aplica acest clasificator pe o problemă de clasificare binară a unei mulţimi de puncte 2D.

2. Fundamente teoretice

AdaBoost, scurt 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 munca lor [2]. În acest laborator vom separa puncte 2D în două clase diferite, clasa punctelor fiind dată de culoarea lor.

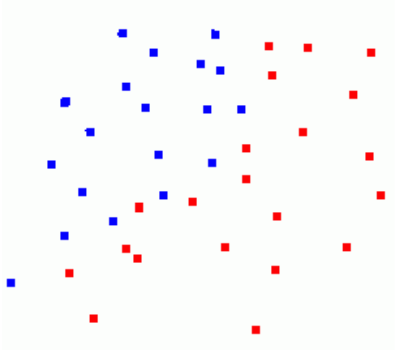

Figure 1 Exemplu de puncte din două clase

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. Vom utiliza un arbore de decizie cu un singur nivel (decision stump) ca 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 matrici 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 w_i=1/n
for t=1:T
  //also returns the weighted training error ε_t:
  [h_t, ε_t] = findWeakLearner(X,y,w)
  α_t = 0.5 ln((1−ε_t)/ε_t)
  s = 0
  for i=1:n
    //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))
    s += w_i
  endfor
  //normalize the weights
  for i=1:n
    w_i ← w_i/s
  endfor
endfor
//returns all the alpha values and the weak learners
return [α, h]

[h_t, ε_t] = findWeakLearner(X,y,w)
best_h = {}
best_err = ∞
for j=1:X.cols
  for threshold=0:img.size //cols or rows
    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
```

Idee 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 greşit 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.

3. Detalii de implementare

Structură sugerată pentru un clasificator slab:

```
struct weaklearner{
  int feature_i;
  int threshold;
  int class_label;
  float error;
  int classify(Mat X){
    if (X.at<float>(feature_i)<threshold)
      return class_label;
    else
      return -class_label;
  }
};
```

Antetul funcţiei care returnează cal 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){
    return 0;
  }
};
```

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

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

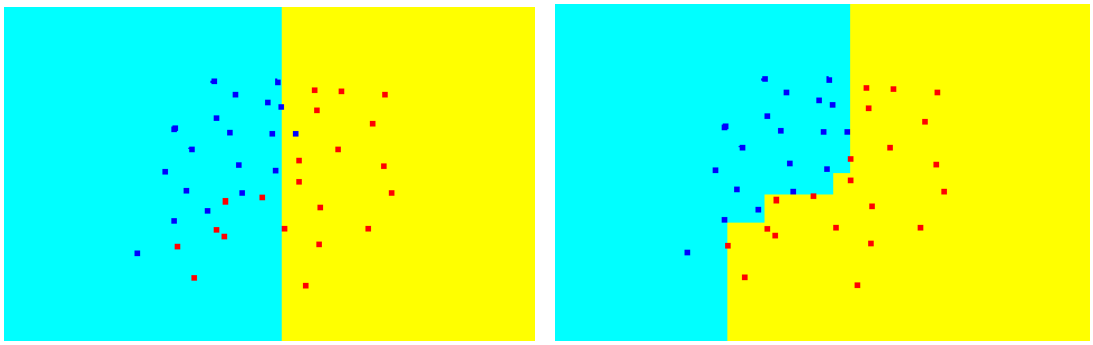

Figure 2. Rezultate obţinute pe points1 cu T=1 (stânga) şi T=13 (dreapta)

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?

5. Referinţe

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

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