Mustererkennung WiSe 12/13
Übung 9

Lutz Freitag, Sebastian Kürten

2 Aufgabe 2: AdaBoost

2.1 Code

2.1.1 a2.m

- Einlesen eines 64x64 Kreisbildes als Eingabe und konvertieren in Schwarz/Weiß
- Generieren von $L = 500$ Klassifikatoren
- Selektieren von $M = 100$ Klassifikatoren mit AdaBoost
- Der Algorithmus gliedert sich in drei Teilschritte wie in der Vorlesung gesehen bzw. im Buch von Prof. Rojas. beschrieben.
- Zeichnen der Eingabegeraden bzw. der von AdaBoost selektierten Geraden. Dabei Darstellung von Gewicht durch Linienbreite und Darstellung der „Richtung" in der AdaBoost die Gerade verwendet, also wierum der Raum durch eine Gerade getrennt wird durch die Farbe. Die Darstellung erfolgt dabei in einem vergrößerten Bild (grauer Bereich) um die ausßerhalb des Eingabebildes liegenden Geraden darstellen zu können.
- Abspeichern der Klassifikatoren und Gewichte zur späteren Weiterverwendung.

```
1  % read image
2  I = imread('a2input64.png');
3  % black and white image
4  % black = 1, white = 2
5  B = (I > 127) + 1;
6
7  % create a feature matrix
8  [h, w] = size(B);
9  data = zeros(h*w, 3);
10 for i = 1:h % row index
11     for j = 1:w % col index
12         % index within data matrix
13         idx = (i-1) * w + j;
14         % coordinates with origin at center
15         x = i - w / 2;
16         y = j - h / 2;
17         % append feature
18         data(idx,:) =  [x y B(i, j)];
19     end
20 end
21
22 % generate a number of linear classifiers as lines in polar coordinates
23 L = 500; % number of classifiers to generate
24 lines = zeros(L, 2);
25 diag = sqrt(h^2+w^2);
26 maxR = diag/2;
27 for c = 1:L
```

```
28      r = rand() * maxR;
29      theta = rand() * 2 * pi;
30      lines(c,:) = [r, theta];
31  end
32  linescopy = lines;
33
34  % initialize weights
35  N = size(data, 1);
36  weights = repmat(1/N, N, 1);
37
38  results = [];
39
40  tic
41  M = 100; % number of classifiers to select
42  for i = 1:M % for each classfier to select
43
44      % lecture's pseudocode: step 1)
45      % select Km which minimizes weighted error
46      best = 0;
47      best_we = inf;
48      % best_dir == 1: use in this direction
49      % best_dir == 0: use in opposite direction
50      best_dir = 1;
51      for m = 1:size(lines, 1) % for each available classifier
52          line = lines(m,:);
53          r = line(1);
54          theta = line(2);
55
56          features = data(:,1:end-1);
57          labels = data(:,end);
58          predictions = hessian_classify(theta, r, 2, 1, features);
59          successIndices = find(predictions == labels);
60          failureIndices = find(predictions != labels);
61          wc = sum(weights(successIndices));
62          we = sum(weights(failureIndices));
63          % pick classifier direction according to wc/we
64          dir = we < wc;
65          uwe = min(we, wc);
66          if uwe < best_we
67              best = m;
68              best_we = uwe;
69              best_dir = dir;
70          end
71      end
72      if best == 0
73          % no classifier can be found
74          break;
75      end
76      m = best;
77      we = best_we;
78
79      % lecture's pseudocode: step 2)
80      % set alpha_m
81      w = sum(weights);
82      em = we / w;
83      alpha_m = 0.5*log((1-em)/em);
84      if (!best_dir)
85          alpha_m = -alpha_m;
86      end
87
88      % lecture's pseudocode: step 3)
89      % update weights
90      line = lines(m,:);
91      r = line(1);
92      theta = line(2);
93
```

```
94      features = data(:,1:end-1);
95      labels = data(:,end);
96      label1 = 2;
97      label2 = 1;
98      if (!best_dir)
99          label1 = 1;
100         label2 = 2;
101     end
102     predictions = hessian_classify(theta, r, label1, label2, features);
103     successIndices = find(predictions == labels);
104     failureIndices = find(predictions != labels);
105     weights(failureIndices) = weights(failureIndices) * sqrt((1-em) / em);
106     weights(successIndices) = weights(successIndices) * sqrt(em / (1-em));
107
108     line = lines(m, :);
109     lines = [lines(1:m-1,:); lines(m+1:end,:)];
110
111     results = [results; alpha_m line];
112
113     [i, alpha_m, line, best_dir]
114 end
115 toc
116
117 % results contains alpha_m weights and parameters of the chosen lines
118 results
119
120 [h, w] = size(I);
121 A = repmat(200, size(I) * 3);
122 A(h:h+h-1,w:w+w-1) = I;
123 A = uint8(A);
124
125 %plotLines(A, linescopy(:,1), linescopy(:,2), repmat(0.5, ...
        size(linescopy, 1), 1), 'a2all.png');
126 plotLines(A, results(:,2), results(:,3), results(:,1) * 10, 'a2result.png');
127
128 save('results.mat', 'results')
```

2.1.2 hessian_classify.m

Klassifizierer für lineare Trennung mit Geraden in hessischer Normalform.

Die Funktion arbeit auf einer Featurematrix und klassifiziert also in einem Schritt mehrere Eingabevektoren. Zur Klassifizierung eines Vektors wird dieser auf den Normalenvektor der Geraden projiziert, und die Länge dieser Projektion bestimmt. Ist die Länge der Projektion kleiner als der Radius der Definition der entsprechenden Gerade in hessischer Normalform, dann liegt der Vektor auf der einen Seite der Geraden, sonst auf der anderen. Dementsprechend werden die Klassen zugeordnet.

```
1  % Klassifizieren alle Daten in 'features'. Trenne linear in class1 und ...
       class2
2  function predictions = hessian_classify(theta, r, class1, class2, features)
3      % normale zu der Geradengleichung
4      normal = [cos(theta); sin(theta)];
5      % Länge der Projektion der Daten auf die Normale
6      projections = (features*normal);
7      % klassifiziere durch Vergleich mit dem Radius
8      class1idx = find(projections < r);
9      class2idx = find(1-(projections < r));
10     predictions = zeros(size(features,1), 1);
11     predictions(class1idx) = class1;
12     predictions(class2idx) = class2;
13 end
```

2.1.3 plotLines.m

Plotten einer Liste von Geraden mit separat spezifizierten Linienbreiten auf ein gegebenes Bild. Die Linienbreite steht dabei für das von AdaBoost zugewiesene Gewicht des zugehörigen Klassifikators. Geraden, deren zugehöriger Klassifikator von AdaBoost mit negativem Gewicht ausgestattet wurden, werden rot dargestellt, die mit positivem Gewicht blau.

```
% plot some lines specified in hessian normal form on an image and store it
% in a file.
function plotLines(I, rs, thetas, widths, filename)
    [h, w] = size(I);
    imshow(I);
    hold on;

    for i = 1:size(rs,1)
        r = rs(i); % radius
        t = thetas(i); % winkel
        lw = widths(i); % line widtdh

        % Länge des zu zeichnenden Geradensegments
        ll = sqrt(h^2+w^2);
        % Normalenvektor der Geraden
        n = [cos(t); sin(t)];
        % Richtungsvektor der Geraden
        v = [n(2); -n(1)];
        % Länge des Richtungsvektors
        l = norm(v);

        % Ankerpunkt
        m = [r * cos(t);  r * sin(t)];
        % zwei Punkte p und q auf der Geraden bestimmen
        p = m - (ll / 2 / l) * v;
        q = m + (ll / 2 / l) * v;
        % zeichnen
        if lw > 0
            plot([w/2 + p(1), w/2 + q(1)], [h/2 - p(2), h/2 - q(2)], ...
                '-b', 'LineWidth', lw);
        else
            plot([w/2 + p(1), w/2 + q(1)], [h/2 - p(2), h/2 - q(2)], ...
                '-r', 'LineWidth', -lw);
        end
    end;

    print(filename);
    hold off;
end;
```

2.1.4 a2test.m

Mit dem folgenden Code werden die Ergebnisse von AdaBoost (die ausgewählten Klassifikatoren mit ihren Gewichten) wieder eingelesen und die Eingabedaten werden klassifiziert. Es ergibt sich eine Erkennungsrate von 98.83% mit der oben genannten Konfiguration.

```
% read image
I = imread('a2input64.png');
% black and white image
% black = 1, white = 2
B = (I > 127) + 1;

% create a feature matrix
```

```
[h, w] = size(B);
data = zeros(h*w, 3);
for i = 1:h % row index
    for j = 1:w % col index
        % index within data matrix
        idx = (i-1) * w + j;
        % coordinates with origin at center
        x = i - w / 2;
        y = j - h / 2;
        % append feature
        data(idx,:) =  [x y B(i, j)];
    end
end

% load classifiers with weights previously generated with AdaBoost
load('results.mat')

features = data(:,1:end-1);
labels = data(:,end);
predictions = zeros(size(features, 1), 1);

% cumulate weighted predictions
for row = results'
    alpha_m = row(1);
    r = row(2);
    theta = row(3);
    % hier übergeben wir als label jetzt nicht mehr 2 und 1, sondern -1 ... und 1,
    % sodass wir später aufgrund von >0 und <0 klassifizieren können.
    predictions += alpha_m * hessian_classify(theta, r, -1, 1, features);
end

% compare with labels
predictions(predictions >= 0) = 1;
predictions(predictions < 0) = 2;

% calculate success rate
successRate = sum(predictions == labels) / size(labels, 1)
```

2.2 Plots

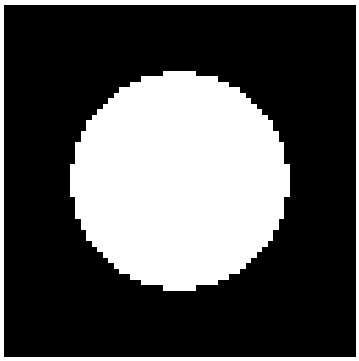

Abbildung 1: Eingabebild

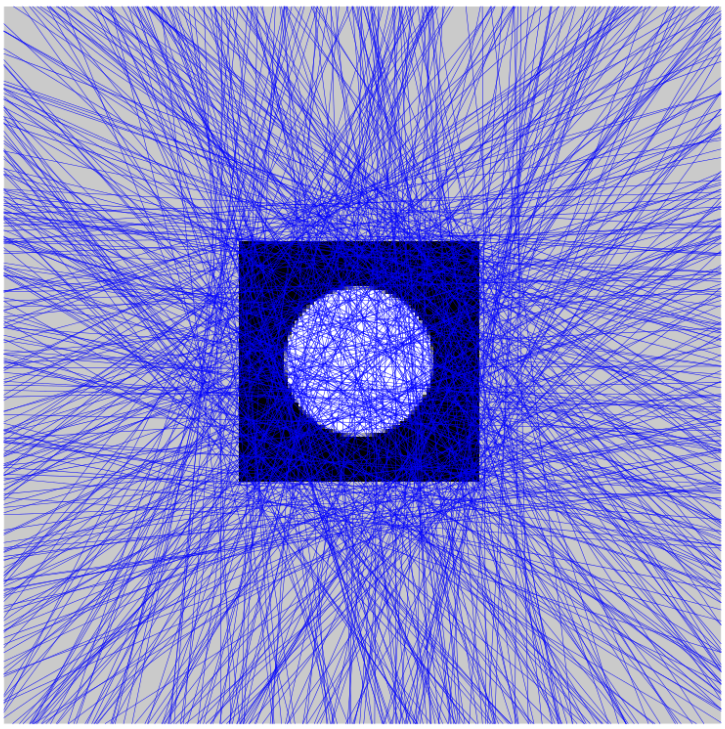

Abbildung 2: Verwendete Klassifikatoren als Eingabe für AdaBoost

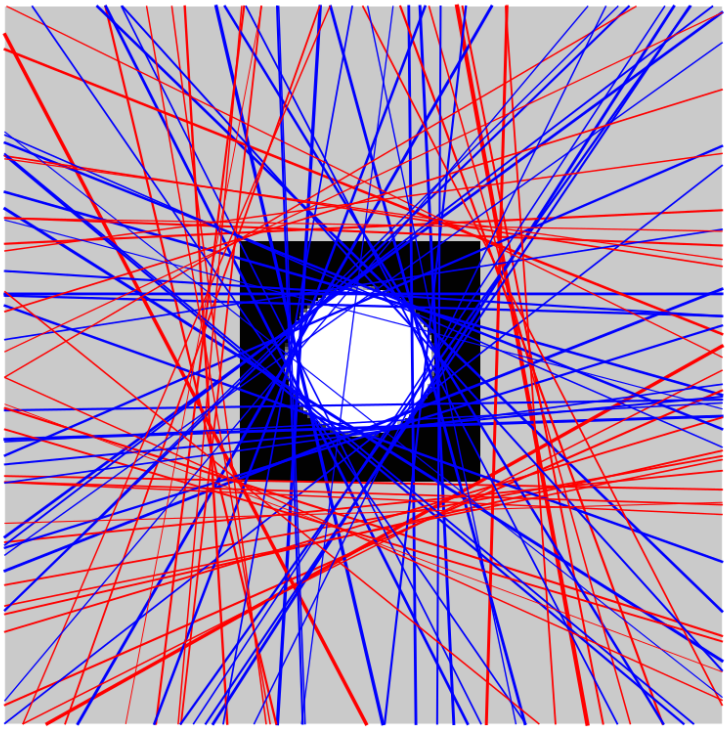

Abbildung 3: von AdaBoost seleketierte Klassifikatoren