**Mustererkennung WiSe 12/13**
**Übung 6**

Lutz Freitag, Sebastian Kürten

# 1 Aufgabe 1: PCA

## 1.1 Teil 1: PCA und Fisher

Wir machen die PCA einmal zu Beginn des Programms mit Hilfe der Trainingsdaten und transformieren sowohl Trainings- als auch Testdaten in den neuen Raum.

Dann trennen wir die Ziffern paarweise mit Hilfe von Fisher's Diskriminante und schauen nach, wie viele Datensätze korrekt klassifiziert wurden im Vergleich zur paarweisen Klassifizierung, die auf den Originaldaten ausgeführt wird. Wir testen mit $0, 1, ..., 6$ weggelassenen Dimensionen und stellen fest:

- beim Weglassen von 0 Dimensionen bleibt die Erkennungsrate gleich
- beim Weglassen von einer Dimension wird die Erkennungsrate insgesamt besser
- beim Weglassen von mehr Dimension wird die Erkennungsrate allmählich schlechter

Der Vergleich erfolgt durch Subtrahieren der Matrizen in der die Anzahl der paarweise korrekt klassifizierten Datensätze erfasst sind.

### 1.1.1 Code

```
1  trainingData = load("-ascii", "pendigits-training.txt");
2  testingData = load("-ascii", "pendigits-testing.txt");
3
4  % Mittelwert der Daten berechnden
5  move = [mean(trainingData(:,1:end-1)) 0];
6  % und Daten zum Ursprung verschieben
7  trainingData = trainingData - repmat(move, size(trainingData, 1), 1);
8  testingData = testingData - repmat(move, size(testingData, 1), 1);
9
10 % features von den labels trennen
11 featuresTraining = trainingData(:,1:end - 1);
12 labelsTraining = trainingData(:, end);
13
14 featuresTesting = testingData(:,1:end - 1);
15 labelsTesting = testingData(:, end);
16
17 % compute mu and covanriance matrix
18 [mu, cov] = gauss(featuresTraining);
19
20 % eigenvektoren und eigenwerte berechnen
21 % eigenvektoren stehen in den spalten
22 % die wichtigste komponente steht ganz rechts
23 [v, lambda] = eig(cov);
24
25
26
```

```
27 % Daten auf die Eigenvektoren projizieren
28 trainingPCA = [featuresTraining * v, labelsTraining];
29 testingPCA = [featuresTesting * v, labelsTesting];
30
31 % für je zwei Ziffern paarweise linear trennen mit
32 % Fisher's Diskriminante und die Anzahl der korrekt / falsch
33 % klassifizierten Daten merken
34 % --> hier mit den Originaldaten als Vergleichswert
35 [correctNormal, wrongNormal] = dofisher(trainingData, testingData);
36
37 % anzahl der zu entfernden Dimensionen
38 for remove = 0:6
39     % die unwichtigsten Hauptkomponenten wegwerfen
40     trainingPCAn = trainingPCA(:,remove + 1:end);
41     testingPCAn = testingPCA(:,remove + 1:end);
42
43     % Nochmal paarweise Fisher.
44     % --> hier mit den dimensionsreduzierten Daten nach PCA
45     [correctPCA, wrongPCA] = dofisher(trainingPCAn, testingPCAn);
46
47     % Anzahl der entfernten Dimensionen ausgeben
48     remove
49     % Differenz der korrekt klassifizieren Daten ausrechnen,
50     % als Vergleich zur Klassifzierung auf den Originaldaten
51     correctPCA - correctNormal
52     total = sum((correctPCA - correctNormal)(:))
53 end
```

### 1.1.2 Ausgabe

```
remove = 0
ans =

   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0

total = 0
remove =  1
ans =

    0    0    0    0    0   -3    1    1    4    0
    0    0    2    0    1   -1    0    0    0    1
    0    2    0    0    0    1    0    0    0    0
   -1    0    0    0    0    1    0    0    0    4
    0    1    0    0    0    0    0   10    0    2
   -3   -1    0    1    0    0   -2    3   -4    4
    1    0    0    0    0   -2    0    2    0    0
    1    0    0    0   10    3    2    0    0   -5
    4    0    0    0    0   -4    0    0    0    0
    0    1    0    4    2    4    0   -5    0    0
```

```
total =  42
remove =  2
ans =

   0   0   0   0   0  -3   1   2   4   0
   0   0   2   0   1  -3   0   4   0   1
   0   2   0   0   0   1   0   1   0   0
  -1   0   0   0   0  -2   0  -7   1   3
   0   1   0   0   0  -4   0   2   0   2
  -3  -3   0  -2  -4   0  -1  -2  -7   2
   1   0   0   0   0  -1   0   2   1   0
   2   4   1  -7   2  -2   2   0   0  -4
   4   0   0   1   0  -7   1   0   0   0
   0   1   0   3   2   2   0  -4   0   0

total = -8
remove =  3
ans =

    0    0    1    0    0    1    0    4    4    1
    0    0  -12    1    1   -5   -1   -4    0    1
    4  -12    0    0    0    2    0    1    0    0
    8    1   -1    0   -2   -1    0   -7    1    4
    0    1    0   -2    0   -3    0   -2    0    2
    1   -5    0   -1   -3    0   -7    0   -8   -2
    0   -1    0    0    0   -7    0    2    1    0
    4   -4    1   -7   -2    0    2    0    0   -4
    4    0    0    1    0   -8    1    0    0    0
    1    1    0    4    2   -2    0   -4    0    0

total = -54
remove =  4
ans =

    0    0    1    0    0    0    0    4    2    1
    0    0  -24    2    1   -6    2   -4    0    1
    4  -24    0    0    0    2    0    1    0    0
    8    2   -1    0   -1   -5    0  -29    2   -3
    0    1    0   -1    0   -2    0   -5    0    2
    0   -6    0   -5   -2    0   -5    1   -8    3
    0    2    0    0    0   -5    0    2    1    0
    4   -4    1  -29   -5    1    2    0    0  -10
    2    0    0    2    0   -8    1    0    0    0
    1    1    0   -3    2    3    0  -10    0    0

total = -140
remove =  5
ans =

    0    0   -1    0   -1    1    1    8    3    1
    0    0  -23   -1    1  -16    1   -4    0    0
    7  -23    0    0    0    2    0   -1    0    0
```

```
    8   -1   -1    0   -3  -11   -1  -29    2    4
   -1    1    0   -3    0  -10    0   -4    0    2
    1  -16    0  -11  -10    0   -6    6   -5    1
    1    1    0   -1    0   -6    0    2    1    0
    8   -4   -1  -29   -4    6    2    0   -1   -9
    3    0    0    2    0   -5    1   -1    0    0
    1    0    0    4    2    1    0   -9    0    0

total = -167
remove =  6
ans =

    0   -1   -2    0    0    2    1   14   -3    1
   -1    0  -27   -2    1  -14    1    0   -1    0
    8  -27    0    0    0    2    0   -2    0    0
    8   -2   -1    0   -2  -11   -1  -31    1    3
    0    1    0   -2    0  -10    0   -7    0    2
    2  -14    1  -11  -10    0   -6    3   -8    1
    1    1    0   -1    0   -6    0    2    0    0
   14    0   -2  -31   -7    3    2    0   -1  -18
   -3   -1    0    1    0   -8    0   -1    0    0
    1    0    0    3    2    1    0  -18    0    0

total = -210
```

## 1.2 Teil 2: PCA für die Ziffer 7 mit Darstellung

Hier haben wir die Hauptkomponenten für die Trainingsziffern mit Label 7 berechnet und diese grafisch dargestellt.

### 1.2.1 Code

```
1
2 % ausgabe der Haupkomponente für die Ziffer 7
3 trainingData = load("-ascii", "pendigits-training.txt");
4 featuresDigit = trainingData(trainingData(:,17) == 7,:);
5 [muDigit, covDigit] = gauss(featuresDigit);
6 [vDigit, lambdaDigit] = eig(covDigit);
7 hold on;
8 for mainComponent = vDigit'
9     mainComponent
10    plot(mainComponent);
11 end
12 hold off;
13
14 print("a1a.png")
15
16 pause
```

### 1.2.2 Ausgabe

siehe Abbildung 1

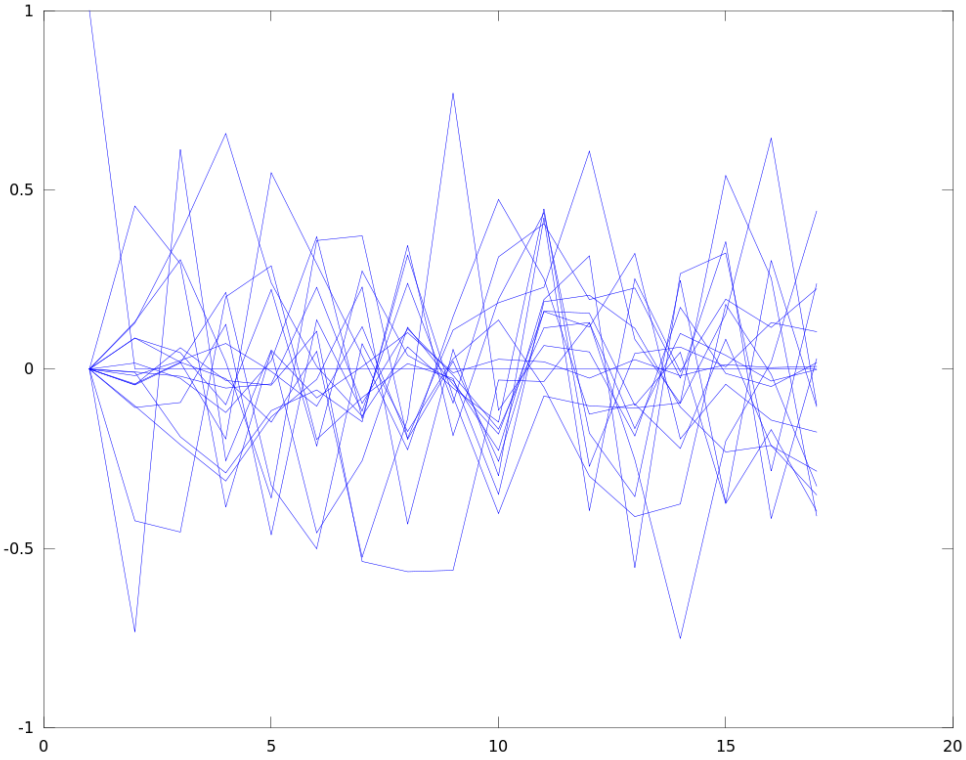


Figure 1: Darstellung der Hauptkomponenten für 7

# 2 Aufgabe 2: Perzeptron-Lernen

## 2.1 Code

### 2.1.1 a2.m

```
% zufälliger Vektor
w = rand([1, 4]) * 2 - 1;

% Trainingsdaten generieren
N = 10000;
X = 2 * rand([N, 3]) - 1;
X = [ones(N, 1) X];

% Labels bestimmen
y = (X * w') >= 0;

% mit Perzeptron lernen
pw = perzeptron(X, y);

% Vorhersage mit dem gelernten w
prediction = (X * pw') >= 0;

% Vorhersage bewerten
nwrong = sum(prediction != y);
relativeError = nwrong / N

w / norm(w)
pw / norm(pw)
```

### 2.1.2 perzeptron.m

```
function bestW = perzeptron(X, y)
    len = size(X, 2);
    w = rand([1, len]) * 2 - 1;
    w = w / norm (w);

    maxIterations = log10(size(X, 1)) * 50;
    % Zähler für Iterationen als zusätzliches Abbruchkriterium
    iteration = 1;
    % beste Lösung merken
    bestNWrong = Inf;
    while true && iteration <= maxIterations
        iteration ++;
        % vorhersage ausrechnen
        prediction = X * w' >= 0;
        % falsch klassifizierte daten finden
        wrong = find(prediction != y);
        % wenn alles richtig klassifiziert -> ende
        nwrong = size(wrong, 1);
        if nwrong < bestNWrong
            bestNWrong = nwrong;
            bestW = w;
        end;
        if nwrong == 0
            break
        end
        % ansonsten, wähle zufällige einen falsch
        % klassifizierten vektor aus
        wk = randint(1,1, [1, nwrong]);
```

```
29          % den index des wk-ten falschen Vektors bestimmen
30          k = wrong(wk);
31
32          % hole den k-ten Vektor
33          x = X(k,:);
34          if y(k)
35              % x ist aus P
36              w = w + x;
37          else
38              % x ist aus N
39              w = w - x;
40          end
41      end
42  end
```

## 2.2 Ausgabe

Wir sehen hier die Fehlerrate, sowie den zu Beginn generierten Vektor und den gelernten Vektor, beide normiert zum Vergleich.

```
relativeError =  0.0019000
ans =

  -0.636667   0.450244  -0.035083   0.625064

ans =

  -0.642846   0.461415  -0.033900   0.610489
```

# 3 Aufgabe 3: Kantenerkennung

## 3.1 Code

### 3.1.1 a3.m

```
% alle möglichen 3x3 Bilder generieren
pixels = binary_permutations(9);

% Bilder labeln
ys = [];
for data = pixels'
    ys(end + 1) = label(data);
end
ys = ys';

% Gewichte lernen
w = perzeptron(pixels, ys);

% Bild laden
I = double(imread('lena.png'));
% nach schwarz-weiß konvertieren
BW = I > mean(I(:));

% Kantenbild berechnen
O = [];
[hei, wid] = size(BW);
% alle Teilbilder ansehen
for y = 2:hei-1
    for x = 2:wid-1
        % Teilbildmatrix holen
        M = BW(y-1:y+1,x-1:x+1);
        % in Vektor umwandeln
        Mpixels = reshape(M, 1, 9);
        % klassifizieren
        O(y-1, x-1) = Mpixels * w';
    end
end
% Bild abspeichern
imwrite(O, "edges.png");
```

### 3.1.2 binarypermutations.m

```
function c = binary_permutations(k)
    k = 9;
    n = 2^k;
    a = arrayfun(@(i) bitget(i, [1:k]), 0:n-1, "UniformOutput", false);
    b = cell2mat(a);
    c = reshape(b, 9, n)';
end
```

### 3.1.3 label.m

```
function r = label(pixels)
    r = pixels(5) == 0 && any(pixels);
end
```

### 3.2 Ausgabe

Figure 2: Eingabebild

Figure 3: Kantenbild

# A Alte Funktionen

## A.1 gauss.m

```
% Mittelwert und Kovarianzmatrix bestimmen
function [mu, cov] = gauss (samples)
        % compute average
        mu = mean(samples);

    s = size(samples, 2);
    % compute covariance matrix
    cov = zeros(s, s);
    for sample = samples'
        cov += (sample - mu') * (sample' - mu);
    end

    cov = cov / size(samples, 1);
end;
```

## A.2 bayes.m

```
function [probA, probB, class] = bayes(x, muA1, muB1, covA1, covB1)
    % Warscheinlichkeiten ausrechnen
    probA = 1 / (sqrt(2*pi) * covA1) * e^((-1/2) * (x-muA1)^2);
    probB = 1 / (sqrt(2*pi) * covB1) * e^((-1/2) * (x-muB1)^2);
    %Wähle die Warscheinlichere Klassennummer
    class = 2;
    if (probA > probB) % hier kommt die Asymmetrie her
        class = 1;
    end
end
```

## A.3 dofisher.m

```
function correct, wrong = dofisher(trainingData, testingData)
    mus = {};
    covariances = {};

    for digit = 0:9
        % select all samples labeled with 'digit'
        samples = trainingData(trainingData(:,end) == digit,:)(:,1:end - 1);
        % compute mu and covanriance matrix
        [mu, cov] = gauss(samples);
        % store for later usage
        mus{digit + 1} = mu;
        covariances{digit + 1} = cov;
    end

    % fisher für zwei klassen ausführen
    %[ncorrect, nwrong] = fisher(mus, covariances, testingData, 0, 1)

    % fisher für alle paare von zwei klassen ausführen
    split_long_rows(false);
    output_max_field_width(2);
    n = 9;
    [correct, wrong] = arrayfun(@(i,j) fisher(mus, covariances, ...
        testingData, i, j), \
```

```
23          repmat([0:n], n+1, 1), repmat([0:n], n+1, 1)');
24  end
```

## A.4 fisher.m

```
1  function [ncorrect, nwrong] = fisher(mus, covariances, testingData, ...
       digitA, digitB)
2      % ignoriere den Fall, dass die Funktion mit digitA = digitB ...
           aufgerufen wird.
3      if (digitA == digitB)
4          ncorrect = 0;
5          nwrong = 0;
6          return;
7      end;
8      % Eingabe sind Zeilenvektoren!
9      testA = testingData(testingData(:,end) == digitA,:)(:,1:end - 1);
10     testB = testingData(testingData(:,end) == digitB,:)(:,1:end - 1);
11     % mus holen
12     muA = mus{digitA + 1};
13     muB = mus{digitB + 1};
14     % Kovarianzen holen
15     covA = covariances{digitA + 1};
16     covB = covariances{digitB + 1};
17     % Projektionsrichtung ausrechnen
18     a = (muA - muB) * (covA + covB)^(-1);
19     % eindimensionale Gaussverteilung bestimmen
20     muA1 = a * muA';
21     muB1 = a * muB';
22     covA1 = a*covA*a';
23     covB1 = a*covB*a';
24     % Projiziere Testdaten auf a
25     pA = testA * a';
26     pB = testB * a';
27     % Wahrscheinlichkeiten nach Bayes
28     [probsA1, probsB1, classes1] = arrayfun(@(x) \
29             bayes(x, muA1, muB1, covA1, covB1), pA);
30     [probsA2, probsB2, classes2] = arrayfun(@(x) \
31             bayes(x, muA1, muB1, covA1, covB1), pB);
32     ncorrect = sum(classes1 == 1) + sum(classes2 == 2);
33     nwrong = sum(classes1 == 2) + sum(classes2 == 1);
34 end
```