Uebungsblatt 2
„Mustererkennung“

J. Cavojska, N. Lehmann

05.05.2015

1 Aufgabe 1 - KNN

Schreiben Sie in matlab oder octave ein Script, das die Datenstze aus chickwts_testing.csv anhand der Datenstze aus chickwts_training.csv mit dem KNN-Algorithmus klassifiziert und die Konfusionsmatrix und Klassifikationsgte (korrekt klassifizierte Individuen geteilt durch Gesamtzahl der Individuen) ausgibt. Geben Sie die Konfusionsmatrix und Klassifikationsgte jeweils fr K = 1, 3 und 5 an.

Notiz: Fuer die Aufg. 1 haben wir Trainingsdaten verwendet, von denen duplikate Eintraege vorher entfernt wurden (Huehner, die das gleiche Gewicht UND die gleiche Futterklasse hatten).

1.1 Code fuer Klassifikator, Konfusionsmatrix und Klassifikationsguete

```
% Spalte 1 = Huhn-ID, Spalte 2 = Gewicht, Spalte 3 = Futterklasse

A = load('chickwts_training.csv');
A_Sorted = sortrows(A,2); % nach Gewichten sortieren
A_Training = A(:,2:3);
U = unique(A_Training,'rows');
zeilen = size(U,1);
B = load('chickwts_testing.csv');
B_Testing = B(:,2:3);
```

```
10
11  alle_k = [1 3 5];
12  for k=alle_k
13    fprintf('k = %i\n',k)
14    dists = zeros(2*k - 1, 2); %  (2*k-1)x2-Matrix mit den 5 Kandidat-Nachbarn
15    % um unser Eingabehuhn h herum. 3 von diesen 5 Nachbarn sind die 3 nearest
16    % neighbors unseres Huhns. 1. Spalte enthaelt die Distanzen, 2. Spalte ↩
          enthaelt
17    % die Klassen dieser Nachbarn.
18
19    C = []; % Ergebnismatrix
20    for zeilenindex_testdaten = 1:size(B_Testing,1)
21      h = B_Testing(zeilenindex_testdaten, 1); % unser Testhuhn
22      % Huhn 'treffer' finden, das unserem Huhn h am naechsten ist:
23      treffer = -1;
24      trefferZeile = -1; % Zeile, in welcher wir das 'treffer'-Huhn gefunden ↩
            haben
25      if h < U(1,1)
26        treffer = U(1,2);
27        trefferZeile = 1;
28      elseif h > U(zeilen,1)
29        treffer = U(zeilen,2);
30        trefferZeile = zeilen;
31      else
32
33        for z = 1:zeilen
34          if h == U(z, 1)
35            treffer = U(z, 2);
36            trefferZeile = z;
37            break
38          elseif h < U(z,1)
39            dist1 = abs(U(z-1, 1) - h);
40            dist2 = abs(U(z, 1) - h);
41            if dist1 <= dist2
42              treffer = U(z-1,2);
43              trefferZeile = z;
44              break
45            else
46              treffer = U(z,2);
47              trefferZeile = z;
48              break
49            end
50          end
51        end
52      end
53      % Wir haben unseren 'treffer' gefunden. Nun entscheiden wir, welche ↩
            Huehner um ihn herum unsere k naechsten Nachbarn sind.
54      if k == 1
55        most_frequent_neighbor_class = treffer;
56      elseif k > 1
57        offset = 0;
58        if trefferZeile < k
59          offset = k - trefferZeile; % 3-1 = 2
60        elseif trefferZeile > zeilen - k + 1
61          offset = zeilen - k + 1 - trefferZeile; % 10-3+1-10 = -2
62        end
```

```
63        dists = U(trefferZeile-(k-1)+offset:trefferZeile+(k-1)+offset, :); % ↩
              aus U ein Fenster der Laenge 2*k-1 um 'treffer' herum ausschneiden
64        dists = [abs((dists(:,1)) - h), dists(:,2)]; % Gewichte ersetzen durch↩
               Distanzen der Gewichte zu h.
65        dists = sortrows(dists,1); % nach Gewicht-Distanzen sortieren
66        k_neighbors_classes = dists(1:k, 2); % die Klassen der k Nachbarn mit ↩
              den kleinsten Gewicht-Distanzen holen
67        most_frequent_neighbor_class = mode(k_neighbors_classes); % findet die↩
               haeufigste Klasse in k_neighbors_classes
68      end
69
70      % Ergebnismatrix C: 1. Spalte: Gewicht, 2. Spalte: Futterklasse (↩
            Testdaten), 3. Spalte: Futterklasse (Trainingsdaten)
71      tmpVector = [B_Testing(zeilenindex_testdaten, 1), B_Testing(↩
            zeilenindex_testdaten, 2), most_frequent_neighbor_class];
72        C = vertcat(C,tmpVector);
73
74    end % end of for each test chicken
75
76    % Konfusionsmatrix
77    knownClass = C(:, 2);
78    predictedClass = C(:, 3);
79    confusionMatrix = confusionmat(knownClass, predictedClass)
80
81    % Klassifikationsguete
82    alle = size(C, 1);
83    korrekt_vorhergesagt = 0;
84    for z = 1:alle
85      if C(z, 2) == C(z, 3)
86        korrekt_vorhergesagt = korrekt_vorhergesagt + 1;
87      end
88    end
89    Klassifikationsguete = korrekt_vorhergesagt / alle
90  end % end of for k in 1, 3, 5
```

1.1.1 Konfusionsmatrizen und Klassifikationsgueten

Rows: actual classes, Columns: predicted classes

k = 1
confusionMatrix =

3	3	1	0	3	0
5	6	1	0	0	0
2	2	4	3	3	0
0	1	1	6	2	2
3	1	1	4	1	1
0	3	3	3	0	3

Klassifikationsguete = 0.3239

k = 3

confusionMatrix =

7	0	2	0	1	0
6	4	0	0	2	0
3	0	6	2	3	0
1	1	2	7	0	1
4	1	1	4	1	0
0	3	3	3	0	3

Klassifikationsguete = 0.3944

k = 5

confusionMatrix =

7	0	2	0	1	0
6	3	0	1	2	0
2	1	7	1	3	0
0	0	3	7	1	1
2	2	1	5	1	0
0	1	3	3	0	5

Klassifikationsguete = 0.4225

2 Aufgabe 2 - Normalverteilung

Notiz: Fuer Aufgabe 2 haben wir die Originaldaten genommen, ohne die Duplikate vorher zu entfernen.

a) Berechnen Sie die Normalverteilung (Erwartungswert und Varianz) ber den Gewichten jeweils fr alle 6 Futterklassen sowie die AprioriWahrscheinlichkeit fr jede der 6 Futterklassen anhand der Werte aus chickwts_training.csv.

2.1 Code fuer Aufg. 2 a)

```
% Berechnungen fuer die 6 Futtermittel FM0..FM5:

FM0_matrix = A_Training(A_Training(:,2)==0,1);
```

```
4  FM1_matrix = A_Training(A_Training(:,2)==1,1);
5  FM2_matrix = A_Training(A_Training(:,2)==2,1);
6  FM3_matrix = A_Training(A_Training(:,2)==3,1);
7  FM4_matrix = A_Training(A_Training(:,2)==4,1);
8  FM5_matrix = A_Training(A_Training(:,2)==5,1);
9
10 % Erwartungswert/Mittelwert berechnen
11 FM0_mean = mean(FM0_matrix);
12 FM1_mean = mean(FM1_matrix);
13 FM2_mean = mean(FM2_matrix);
14 FM3_mean = mean(FM3_matrix);
15 FM4_mean = mean(FM4_matrix);
16 FM5_mean = mean(FM5_matrix);
17
18 % Varianz
19 FM0_var = var(FM0_matrix);
20 FM1_var = var(FM1_matrix);
21 FM2_var = var(FM2_matrix);
22 FM3_var = var(FM3_matrix);
23 FM4_var = var(FM4_matrix);
24 FM5_var = var(FM5_matrix);
25
26 % Standardabweichungen
27 FM0_std = std(FM0_matrix);
28 FM1_std = std(FM1_matrix);
29 FM2_std = std(FM2_matrix);
30 FM3_std = std(FM3_matrix);
31 FM4_std = std(FM4_matrix);
32 FM5_std = std(FM5_matrix);
33
34 % A Priori Wahrscheinlickeit berechnen
35 FM0_apriori = length(FM0_matrix) / length(A_Training)
36 FM1_apriori = length(FM1_matrix) / length(A_Training)
37 FM2_apriori = length(FM2_matrix) / length(A_Training)
38 FM3_apriori = length(FM3_matrix) / length(A_Training)
39 FM4_apriori = length(FM4_matrix) / length(A_Training)
40 FM5_apriori = length(FM5_matrix) / length(A_Training)
```

2.2 Ergebnisse fuer Aufg. 2 a)

FM0_mean = 165.7200
FM1_mean = 217.9167
FM2_mean = 244.6429
FM3_mean = 325.6000
FM4_mean = 281.8182
FM5_mean = 326.9333

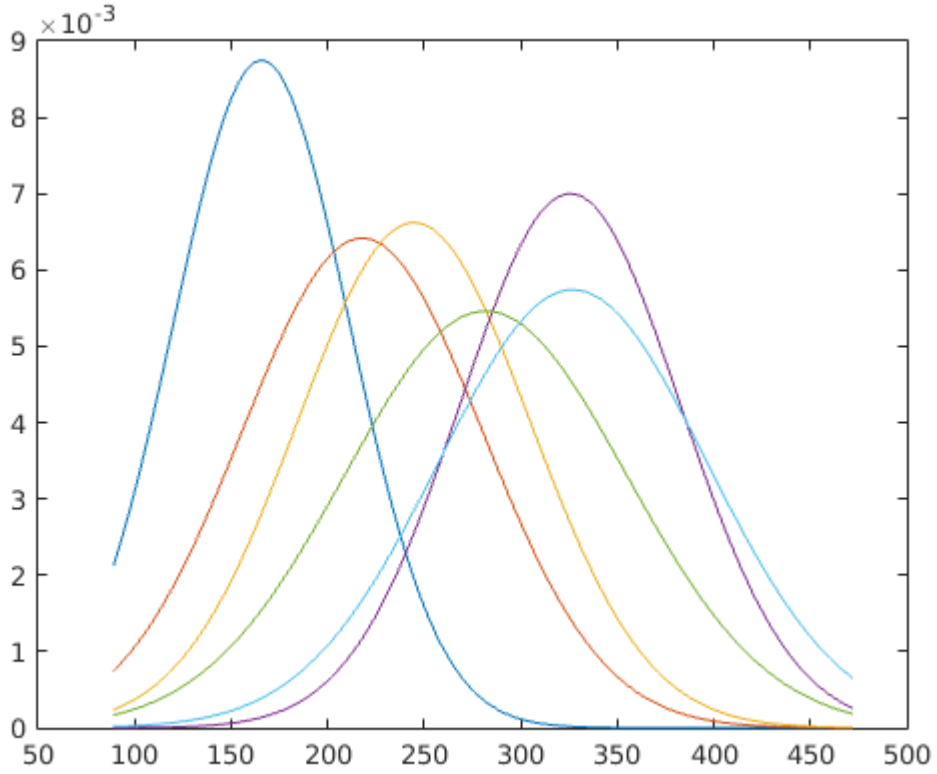
FM0_var = 2.0801e+03
FM1_var = 3.8649e+03
FM2_var = 3.6319e+03

FM3_var = 3.2483e+03
FM4_var = 5.3364e+03
FM5_var = 4.8296e+03

FM0_std = 45.6079
FM1_std = 62.1682
FM2_std = 60.2656
FM3_std = 56.9937
FM4_std = 73.0505
FM5_std = 69.4952

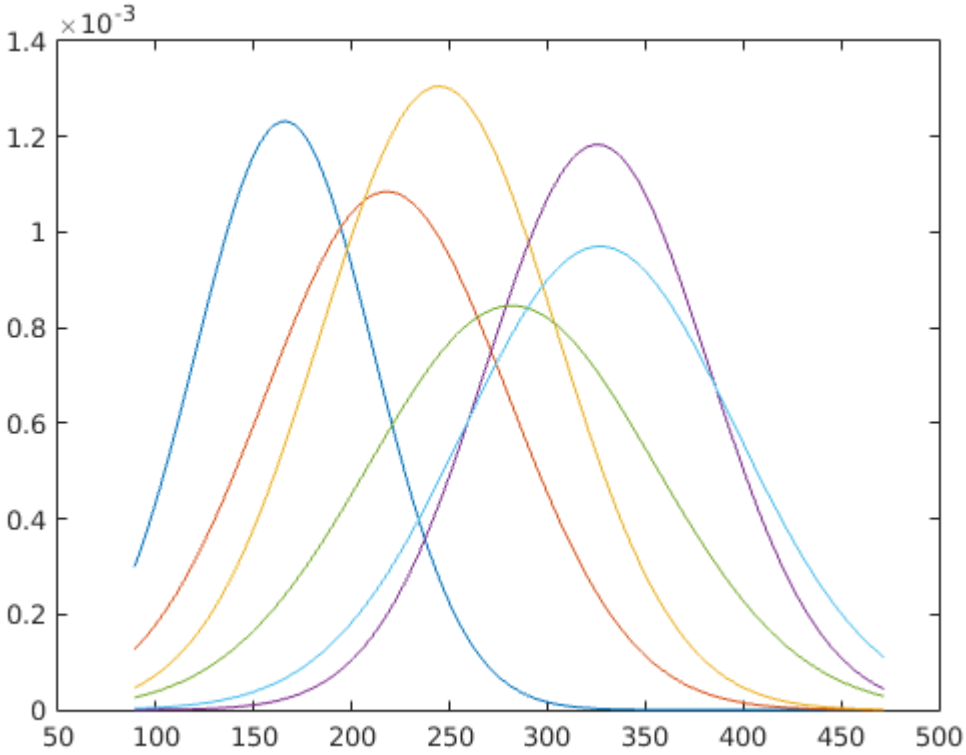
FM0_apriori = 0.1408
FM1_apriori = 0.1690
FM2_apriori = 0.1972
FM3_apriori = 0.1690
FM4_apriori = 0.1549
FM5_apriori = 0.1690

```
\% PDFs plotten (in einer Grafik)\\
X = A_Training(:,1);\\
x = min(X):max(X); \% Abschnitt auf der x-Achse, der geplottet werden soll\\
\\
\% PDFs berechnen\\
\% pdf(Art von Verteilung, Abschnitt auf x-Achse, mean, std)\\
FM0_pdf = pdf('Normal',x,FM0_mean, FM0_std);\\
FM1_pdf = pdf('Normal',x,FM1_mean, FM1_std);\\
FM2_pdf = pdf('Normal',x,FM2_mean, FM2_std);\\
FM3_pdf = pdf('Normal',x,FM3_mean, FM3_std);\\
FM4_pdf = pdf('Normal',x,FM4_mean, FM4_std);\\
FM5_pdf = pdf('Normal',x,FM5_mean, FM5_std);\\
\\
P1 = plot(x,FM0_pdf,x,FM1_pdf,x,FM2_pdf,x,FM3_pdf,x,FM4_pdf,x,FM5_pdf);\\
```



b) Plotten Sie die A-posteriori-Wahrscheinlichkeitsdichtefunktion p(Futterklasse — Gewicht) fr alle 6 Futterklassen zusammen in einem Diagramm. Verwenden Sie dabei die Werte aus a).

```
FM0_aposteriori = FM0_pdf * FM0_apriori;
FM1_aposteriori = FM1_pdf * FM1_apriori;
FM2_aposteriori = FM2_pdf * FM2_apriori;
FM3_aposteriori = FM3_pdf * FM3_apriori;
FM4_aposteriori = FM4_pdf * FM4_apriori;
FM5_aposteriori = FM5_pdf * FM5_apriori;
% PDFs plotten (in einer Grafik)
P2 = plot(x,FM0_aposteriori,x,FM1_aposteriori,x,FM2_aposteriori,x,↩
    FM3_aposteriori,x,FM4_aposteriori,x,FM5_aposteriori);
```



c) Klassifizieren Sie die Hhner in chickwts_testing.csv mit einem Bayes-Klassifikator anhand ihres Gewichtes in die 6 Futterklassen. Verwenden Sie dabei die Werte bzw. Wahrscheinlichkeitsdichtefunktionen aus a) und b). Geben Sie die die Konfusionsmatrix und Klassifikationsgte aus.

```
D = [];
for huhnIndex = 1:size(B_Testing,1)
  h = B_Testing(huhnIndex, 1);
  fm0pre = pdf('Normal', h, FM0_mean, FM0_std) * FM0_apriori;
  fm1pre = pdf('Normal', h, FM1_mean, FM0_std) * FM1_apriori;
  fm2pre = pdf('Normal', h, FM2_mean, FM0_std) * FM2_apriori;
  fm3pre = pdf('Normal', h, FM3_mean, FM0_std) * FM3_apriori;
  fm4pre = pdf('Normal', h, FM4_mean, FM0_std) * FM4_apriori;
  fm5pre = pdf('Normal', h, FM5_mean, FM0_std) * FM5_apriori;

  [maxValue, indexAtMaxValue] = max([fm0pre,fm1pre,fm2pre,fm3pre,fm4pre,↩
      fm5pre]);

```

```
13    if (maxValue == fm0pre)
14      tmpVector = [B_Testing(huhnIndex,1),B_Testing(huhnIndex,2),0];
15      D = vertcat(D,tmpVector);
16    elseif (maxValue == fm1pre)
17      tmpVector = [B_Testing(huhnIndex,1),B_Testing(huhnIndex,2),1];
18      D = vertcat(D,tmpVector);
19    elseif (maxValue == fm2pre)
20      tmpVector = [B_Testing(huhnIndex,1),B_Testing(huhnIndex,2),2];
21      D = vertcat(D,tmpVector);
22    elseif (maxValue == fm3pre)
23      tmpVector = [B_Testing(huhnIndex,1),B_Testing(huhnIndex,2),3];
24      D = vertcat(D,tmpVector);
25    elseif (maxValue == fm4pre)
26      tmpVector = [B_Testing(huhnIndex,1),B_Testing(huhnIndex,2),4];
27      D = vertcat(D,tmpVector);
28    else
29      tmpVector = [B_Testing(huhnIndex,1),B_Testing(huhnIndex,2),5];
30      D = vertcat(D,tmpVector);
31    end
32  end % end of for each h
33
34  % Konfusionsmatrix
35    knownClass = D(:, 2);
36    predictedClass = D(:, 3);
37    confusionMatrix = confusionmat(knownClass, predictedClass)
38
39    % Klassifikationsguete
40    alle = size(D, 1);
41    korrekt_vorhergesagt = 0;
42    for z = 1:alle
43      if D(z, 2) == D(z, 3)
44        korrekt_vorhergesagt = korrekt_vorhergesagt + 1;
45      end
46    end
47    Klassifikationsguete = korrekt_vorhergesagt / alle
```

Konfusionsmatrix und Klassifikationsguete

confusionMatrix =

8	1	1	0	0	0
4	2	5	1	0	0
2	2	7	1	0	2
0	0	1	3	2	6
1	1	4	3	0	2
0	1	2	1	1	7

Klassifikationsguete = 0.3803