

Künstliche Intelligenz - Übung 6

Adrian Neumann (4140810) und Naja von Schmude (4127652)

5. Juni 2009

Aufgabe 1

Wir haben uns für diese Aufgaben eine Digit-Klasse geschrieben, die unter anderem über eine Methode verfügt, um den euklidischen Abstand zu einem anderen Digit zu bestimmen.

Damit kann man dann relativ leicht KNN machen. Wenn man einen Vektor klassifizieren will, rechnet man einfach einmal alle Abstände zu den Vektoren der Trainingsmenge aus und selektiert dann die k kleinsten.

```
/**
 * Gibt ein Array mit den nächsten Nachbarn zurück (die stecken auch in nb
 * drin)
 * @param l Die Menge aus der die nächsten Nachbarn bestimmt werden sollen
 * @param dist Ein Array in das die Distanzen zu allen anderen Vektoren
 * reinkommen. Muss genauso groß sein wie l.
 * @param nb Ein Array in das die Nachbarn reinkommen. Seine Länge bestimmt
 * die Zahl der nächsten Nachbarn
 * @return nb */
private static Vektor[] nearestNeighbours(Vektor d, Vektor[] l, double[] dist, Vektor
[] nb) {
    //wir rechnen einmal alle Abstände aus
    for(int i=0; i<l.length; i++)
        dist[i] = d.dist(l[i]);
    //und selektieren die Objekte mit den kleinsten Abständen
    if (nb.length>1)
        selectMinK(l,dist,nb);
    else {
        //Wenn wir nur das Minimum suchen, brauchen wir nicht den
        //komplizierten Algorithmus anzuwerfen.
        double min=Double.MAX_VALUE;
        int minI=-1;
        for(int i=0; i<dist.length; i++)
            if (dist[i]<min) {
                min=dist[i]; minI=i;
            }
        nb[0]=l[minI];
    }
    return nb;
}

/**
 * @param l Objekte von denen man die k kleinsten selektieren soll
 * @param dist die Gewichte, nach denen die Objekte geordnet sein sollen
 * @param nb das Array, wo die k kleinsten Objekte reinkommen sollen. */
private static void selectMinK(Object[] l, double[] dist, Object[] nb) {
    final int k= nb.length;
    int right=dist.length-1;
    int left=0;

    double tmpD;
    Object tmpV;
    double pivot;
    int i,j;
```

```

/* Wir machen QuickSelect, um in  $O(n)$  die Dinger zu finden
 * Das sieht ein bisschen nach QuickSort aus */
while(right>k && right>=left) {
    pivot = dist[right];

    i=left;
    j=right-1;
    //Quicksortmäßig
    do {
        while(dist[i]<=pivot && i<right) ++i;
        while(dist[j]>=pivot && j>0) --j;
        if (i<j) {
            //sowohl Gewicht, als auch Objekte tauschen
            tmpD = dist[j];
            dist[j]=dist[i];
            dist[i]=tmpD;

            tmpV = l[j];
            l[j]=l[i];
            l[i]=tmpV;
        } else {
            break;
        }
    } while(true);

    if (dist[i]>pivot) {
        tmpD = dist[right];
        dist[right]=dist[i];
        dist[i]=tmpD;

        tmpV = l[right];
        l[right]=l[i];
        l[i]=tmpV;
    }
    if(i>k) { //wenn das pivotelement rechts ist
        right = i-1; //können wir die rechte seite verschieben
    } else { //ansonsten müssen wir in der rechten hälfte nochmal sortieren
        left = i+1;
    }
}
//dann noch fix rüberkopieren und fertig
System.arraycopy(l, 0, nb, 0, k);
}

```

Wenn man die k nächsten Nachbarn hat, braucht man nur noch zu kucken, welche Lbel sie haben und dem zu klassifizierenden Vektor das Label der Mehrheit zu geben.

```

public static int votum(Vektor[] voters, int maxlabel) {
    int[] labels = new int[maxlabel+2];
    int max=-1, maxI=-1;
    int l;
    for(Vektor u:voters) {
        l=u.getLabel()+1;
        labels[l]++;
        if (labels[l]>=max) { //>= ist für unsere Daten besser als >
            max=labels[l]; maxI=l;
        }
    }
    return maxI-1;
}

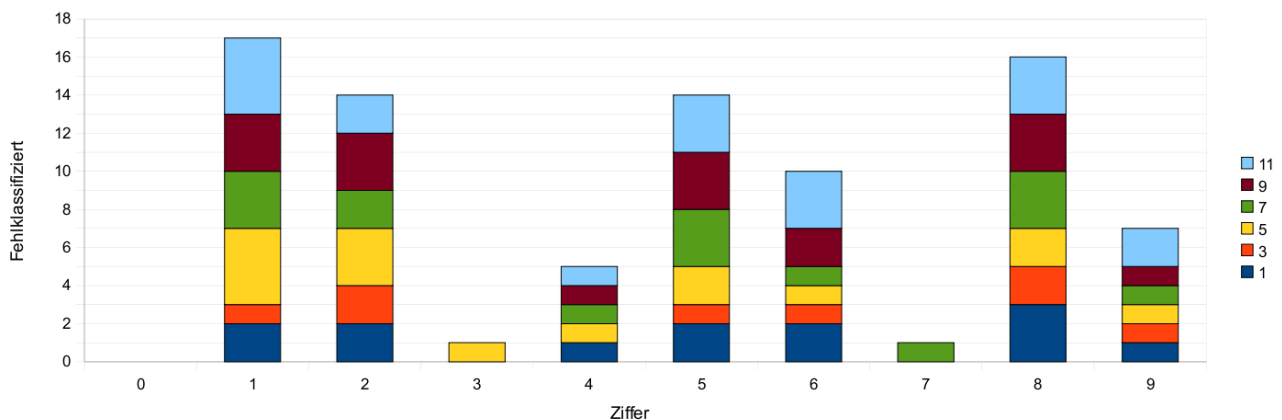
```

Die Klassifizierung ist optimal für $k=3$. Es ergeben sich folgende Fehler:

Zahl	0	1	2	3	4	5	6	7	8	9
Anzahl Testvektoren	32	21	25	23	23	14	13	20	10	18

k	0	1	2	3	4	5	6	7	8	9	Gesamt
1	0	2	2	0	1	2	2	0	3	1	13 (6.53%)
3	0	1	2	0	0	1	1	0	2	1	8 (4.02%)
5	0	4	3	1	1	2	1	0	2	1	15 (7.54%)
7	0	3	2	0	1	3	1	1	3	1	15 (7.54%)
9	0	3	3	0	1	3	2	0	3	1	16 (8.04%)
11	0	4	2	0	1	3	3	0	3	2	18 (9.05%)

Wir haben mal ein Bild mit den akkumulierten Fehlern gemacht, damit man sieht, welche Zahlen besonders schwierig zu klassifizieren waren:



Aufgabe 2

Wenn man schon KNN hat, ist es nicht so schwierig k-Means zu machen. Man wählt sich einfach zufällige Repräsentanten und gibt ihnen temporäre Labels. Dann klassifiziert man die Trainingsmenge mit den Repräsentanten und setzt sie in die Mitte der Vektoren, die ihnen zugeordnet wurden. Das iteriert man so lange, bis sie sich nicht mehr oder nur noch wenig bewegen (das muss allerdings nicht immer eintreten).

```
/**
 * @param k Anzahl der Repräsentanten
 * @param l Zu clusternde Daten
 * @return Repräsentanten */
public static Vektor[] cluster(int k, Vektor[] l) {
    //Wir wählen k zufällige Repräsentanten
    Vektor[] repraesentanten = new Vektor[k];
    for(int i=0; i<k; i++) {
        repraesentanten[i] = Digit.random();
        repraesentanten[i].setLabel(i);
    }

    double moved;
    int[] zahl;
    do {
        moved=0;
        //Platz für die Repräsentanten aus dieser Iteration
        double[][] neuReps = new double[k][Digit.dim];
        zahl = new int[k];

        //für knn
        double[] dists = new double[k];
        Vektor[] nb = new Vektor[1];
    } while (moved > 0.001);
}
```

```

for(Vektor v:l) { //Alle Vektoren mit den Repräsentanten klassifizieren
    // und ihren Mittelpunkt bestimmen
    int label = KNN.classify(v, repraesentanten, dists, nb, k);
    zahl[label]++;
    add(neuReps[label],v.getV()); //zusammenaddieren
}

for(int i=0; i<k; i++) {
    if (zahl[i]!=0) div(neuReps[i],zahl[i]);
    //Die double Arrays werden jetzt noch in Digits umgewandelt
    //und es wird ausgerechnet, wie weit wir gewandert sind
    Digit d = new Digit(neuReps[i],i);
    moved += Math.abs(repraesentanten[i].dist(d));
    repraesentanten[i]=d;
}
} while (moved > (1/k));

//Jetzt labeln wir die Repräsentanten noch mit den Labeln "ihrer"
//Gruppe und geben dabei Statistiken aus, speichern Bilder
double[] dist = new double[l.length];
Vektor[] nb = new Vektor[3];
int i=0;
for(Vektor d : repraesentanten) {
    d.setLabel(KNN.classify(d, l, dist, nb,9));
    String path = "./"+d.getLabel()+"_"+i+".png";
    System.out.printf("Repräsentant für %d in der Datei %s, mit %d zugeordneten Vektoren\n",d.getLabel(),path,zahl[i++]);
    ((Digit) d).savePNG(path);
}
return repraesentanten;
}

```

Wenn man die Repräsentanten hat, kann man sie dann natürlich für ein KNN mit $k = 1$ benutzen, um die Testdaten zu klassifizieren. Die Erkennungsraten sind allerdings deutlich schlechter als bei ordentlichem KNN. Wir haben hier mal ein Bildchen gemacht um den Zusammenhang zwischen Erkennungsrate und Anzahl der Repräsentanten anschaulich darzustellen. Wir haben Mittelwerte über 15 Läufe gemacht:

