

Mustererkennung: Übungsblatt 3

von

Naja v. Schmude (4127652), Lisa Dohrmann (4130066), Adrian Neumann (4140810)

Aufgabe 1

Code

In der Aufgabe sollen per linearer Regression und unter Anwendung des stratified k-fold cross-validation Verfahrens die Erkennungsrate eines Herzinfarktes über der gegebenen Personendatenbank ermittelt werden. Im einfachsten Sinne sucht die lineare Regression eine optimale Gerade durch die Daten, so dass ein neuer Datenpunkt einfach durch „Einsetzen in die Geradengleichung“ klassifiziert werden kann. Da es sich um mehrdimensionale Daten handelt, muss natürlich mit Matrizen und Vektoren gearbeitet werden. Die „Geradengleichung“ sieht hier wie folgt aus:

$$\vec{y} = X \cdot \vec{\beta}$$

X ist dabei unsere Datenmatrix, die wir klassifizieren wollen und $\vec{\beta}$ könnte man mit der Steigung vergleichen, \vec{y} ist dann die Klassifizierung, in dem in Zeile i die Klassifizierung für die Daten in Zeile i der Matrix X steckt. Man muss also zunächst mit der Trainingsmenge, die uns \vec{y} und X vorgibt, das $\vec{\beta}$ berechnen. Um für die Testdaten dann aus X und dem berechneten $\vec{\beta}$ die Klassifizierung zu erhalten. Mathematisch geht das so

$$\vec{\beta} = (X^T X)^{-1} X^T \vec{y}.$$

Wir arbeiten mit dem Matrixpaket *JAMA*, in Java sieht das dann wie folgt aus:

```

1 private Matrix beta(Matrix data, Matrix y) {
2     //damit die Matrix invertierbar sein kann, muss sie quadratisch sein
3     //Darum wird einmal transponiert und dann mit der ursprungsmatrix
      multipliziert.
4     //aus NxM und MxN wird dann NxN
5     Matrix transpose = data.transpose();
6     Matrix square = transpose.times(data);
7     //Diese Matrix sollte jetzt invertierbar sein
8     Matrix inv = square.inverse();
9     //jetzt kann man den Vektor Beta einfach ausrechnen
10    Matrix beta = inv.times(transpose).times(y);
11    return beta;
12 }
```

Mit dem $\vec{\beta}$ kann dann nun für Testdaten die Klassifizierung berechnet werden:

```

1 public Matrix classify(Matrix data, Matrix beta) {
2     Matrix y = data.times(beta);
3     double[][] classifyA = y.toArray();
4     for (int j = 0; j < classifyA.length; j++) {
5         //so ähnlich wie Math.signum, nur dass 0->1 abgebildet wird
6         if (classifyA[j][0] >= 0)
7             classifyA[j][0] = 1;
8         else
9             classifyA[j][0] = -1;
10    }
11    Matrix classify = new Matrix(classifyA);
12    return classify;
13 }
```

Wir müssen dabei noch ein wenig die Klassifizierungen anpassen und auf -1 bzw. 1 mappen, denn wir wollen ja nur eine binäre Aussage treffen und nicht beliebig viele Abstufungen haben.

Interessant ist es nun, wie man geschickt die Daten in Trainings- und Testblöcke aufteilt. Dies geschieht mittels stratified k-fold cross-validation.

Bei dem Verfahren werden die Daten nun in k gleich verteilte Teilmengen unterteilt, die dann jeweils wie folgt validiert werden: Man nimmt $k - 1$ Teilmengen als Trainingsmenge und die übrig bleibende Teilmenge

als Testmenge. Das ganze führt man k mal durch, so dass jede der k Teilmengen einmal Testmenge gewesen ist. Dadurch erhält man k Erkennungsraten, die dann gemittelt werden, was dann die gesamte Erkennungsrate darstellt.

```

1 public double straitfieldKFoldCrossValidation(List<Person> people, int k)
2 {
3     // data enthält hiernach Teilmengen der Größe k, die etwa gleichviele
4     // mit und ohne Herzleiden enthalten
5     List<List<Person>> data = partingData(k, splitInGroups(people));
6
7     // Hier beginnt das eigentliche ...
8     // rate ist die Erkennungsrate über alle Durchläufe addiert
9     double rate = 0.0;
10    for (int i = 0; i < k; i++) { // k mal durchlaufen lassen
11        List<Person> trainingP = new ArrayList<Person>();
12        List<Person> testP = new ArrayList<Person>();
13        Matrix training;
14        Matrix test;
15        // trainingsmenge und testmenge bauen
16        // Eine der Teilmengen aus data wird pro Schleifendurchlauf als
17        // Testmenge genommen
18        // alle anderen zusammen bilden die Trainingsmenge
19        // Da wir das k-mal machen, kommt jede Teilmenge einmal als Testmenge
20        // ran.
21        for (int j = 0; j < k; j++) {
22            if (i == j) {
23                testP = data.get(j);
24            } else {
25                trainingP.addAll(data.get(j));
26            }
27        }
28        test = generateDataMatrix(testP);
29        training = generateDataMatrix(trainingP);
30
31        //Klassifizieren und auswerten
32        //Vektor der Klassifizierungen wird erstellt
33        Matrix y = y(trainingP);
34        //Multipliziert man einen Featurevektor (als Zeilenvektor) mit beta,
35        //bekommt man die Klassifizierung
36        Matrix beta = beta(training, y);
37        //In der Testmatrix stehen alle Daten als Zeilenvektoren drin.
38        //Einmal multiplizieren liefert den Klassifizierungsvektor
39        Matrix classify = classify(test, beta);
40
41        //jetzt überprüfen wir noch, ob die Klassifizierung mit dem Label
42        //übereinstimmt.
43        double[][] classifyA = classify.toArray();
44        int hit = 0;
45        for (int j = 0; j < classifyA.length; j++) {
46            if (testP.get(j).getChd() == classifyA[j][0])
47                hit++;
48        }
49        rate += (double) hit / classifyA.length;
50    }
51    return rate / k;
52 }

```

Die Teilmengen erzeugen wir mit Hilfe der folgenden Methode.

```

1 public List<List<Person>> partingData(int k,
2     Tuple<List<Person>, List<Person>> people) {
3     List<List<Person>> data = new ArrayList<List<Person>>();
4     int anzHD = people.getOne().size() / k; // Anzahl Herzkrankte
5     int anzNHD = people.getTwo().size() / k; // Anzahl nicht Herzkrankter
6     for (int i = 0; i < k; i++) {
7         data.add(new ArrayList<Person>());
8         data.get(i).addAll(
9             people.getOne().subList(i * anzHD, (i + 1) * anzHD));
10        data.get(i).addAll(
11            people.getTwo().subList(i * anzNHD, (i + 1) * anzNHD));
12    }
13 }

```

```
13 |   return data;  
14 | }
```

Dabei ist `people` ein Tupel von zwei Listen. Die erste enthält dabei alle Personen, die das Herzleiden haben, die andere, die Personen ohne. Dadurch, dass man jetzt immer den k -ten Teil jeder Liste zusammenschmeißt, erhält man sowas wie eine Gleichverteilung pro Block. Es wird dann eine Liste von diesen k gleich verteilten Personenlisten zurückgegeben.

Auswertung

Für $k \in [3, 50]$ sind die Erkennungsraten alle relativ konstant zwischen 72% und 75%. Dabei muss noch gesagt werden, dass wir die Daten mit -1 Klassifizieren, wenn die Person kein Herzleiden hat und mit 1 , falls sie krank ist. Als wir probeweise mit 0 und 1 gerechnet haben, so wie es auch in der Datenbank steht, so kamen wir auf Werte, die um die 40% lagen, also deutlich schlechter sind.