Mustererkennung - Übung3

Semjon Kerner, Philip Schmidt, Samuel Gfrörer

2016-11-10

1 Klassifikation mit Normalvergteilungen

1.1 Ausführen des Programms

Das Programm kann folgendermaßen ausgeführt werden:

```
$ ./scripts/run.sh
```

1.2 Implementierung

Der gesamte Code sowie die ausführbare Datei befindet sich im beigefügten Archiv. Der Code wird als Github-Repository verwaltet (siehe https://github.com/EsGeh/pattern-recognition). Um das Programm selbst zu installieren und zu kompilieren, siehe unten.

1.2.1 Ordnerstruktur

```
.
|-- app
|   `-- Main.hs
|-- resource
|   |-- ...
|-- src
|   |-- PatternRecogn
|   |   `-- Gauss.hs
|   `-- ...
```

Der Code ist folgendermaßen aufgeteilt:

1. Ausführbare Date (siehe "./app")

   In der "./app/Main.hs" befindet sich der Code zum Einlesen der Test-Daten.

2. Bibliothek (siehe "./src")

   Hier befindet sich die eigentliche Funktionalität des Klassifizierungsalgorithmus

Im Ordner "./resource" befinden sich die Trainingsdatensätze und der Testdatensatz.

1.2.2 Die Funktionalität des Programms

Das Programm wählt nacheinander alle 2-Tupel der Trainingsdatensätze in "./resource/train.*" und klassifiziert die Testdaten in "./resource/zip.test":

```
33 main :: IO ()
34 main =
35     handleErrors $
36     do
37         mapM_
38             (uncurry4 testWithData . uncurry testParamsFromLabels) $
39             allPairs [3,5,7,8]
40     where
41         handleErrors x =
42             ...
```

Das heißt die Funktion "testWithData" wird z.B. mit den Parametern zweier Dateinamen und den entsprechenden Labels aufgerufen:

```
49 testWithData :: FilePath -> FilePath -> Label -> Label -> ErrT IO ()
50 testWithData trainingFile1 trainingFile2 label1 label2 =
51     do
52         ...
53         [trainingSet1, trainingSet2] <-
54             mapM (readData trainingDataFormat) $
55             [ trainingFile1
56             , trainingFile2
57             ]
58         (inputLabels, inputData) <-
59             prepareInputData (`elem` [fromIntegral label1, fromIntegral label2]) <$>
60             readData inputDataFormat "resource/zip.test"
61         let classificationParam = calcClassificationParams trainingSet1 trainingSet2
62         let classified = classify (label1,label2) classificationParam inputData
63         let result =  calcClassificationQuality (cmap round $ inputLabels) classified
64         liftIO $ putStrLn $
65             descriptionString
66                 trainingSet1
67                 trainingSet2
68                 classificationParam
```

```
69                  inputData
70                  classified
71                  result
```

Die beiden Trainingsdatensätze werden aus den Dateinamen in die Variablen "trainingSet1" und "trainingSet2" geladen. Danach werden die Testdaten in die Variable "inputData" geladen. Die Labels sind für den Testdatensatz bekannt, und werden in der Variablen "inputLabels" gehalten. Die Funktion "calcClassificationParams" berechnet die Parameter für die Klassifizierung. Die Funktion "classify" klassifiziert dann die Testdaten anhand der errechneten Information. Die letzten beiden Befehle berechnen die Qualität der Klassifizierung, und geben sie aus.

Der Code für die Vorberechnung und das Klassifizieren befindet sich in "./src/PatternRecogn/Gauss.hs": Die Vorberechnung werden im folgenden Code-Ausschnitt deutlich:

```haskell
17  data ClassificationParam
18      = ClassificationParam {
19          min1 :: Vector,
20          min2 :: Vector,
21          covariance1 :: Matrix,
22          covariance2 :: Matrix
23      }
24
25  calcClassificationParams :: Matrix -> Matrix -> ClassificationParam
26  calcClassificationParams set1 set2 =
27      ret
28      where
29          ret = ClassificationParam {
30              min1 = average $ toRows set1,
31              min2 = average $ toRows set2,
32              covariance1 = cov_SAFE (min1 ret) set1,
33              covariance2 = cov_SAFE (min2 ret) set2
34          }
```

Der Typ "ClassificationParam" speichert das Ergebnis der Vorberechnung. Die Zeilen der übergebenen Matrizen "set1" und "set2" enthalten die Features jeweils eines Samples. "min1" und "min2" stehen für den Erwartungswert dieser Feature-Vektoren, hier also deren Durchschnitt. Außerdem werden die Kovarianz-Matrizen beider Trainingsdatensätze berechnet.

Die Funktion "classify" klassifiziert die Testdaten anhand der oben beschriebenen "ClassificationParam":

```haskell
72      classify :: (Label, Label) -> ClassificationParam -> Matrix -> Lina.Vector Label
73      classify (labelNeg, labelPos) ClassificationParam{..} =
74          fromList
75          .
```

```haskell
        map classifySingleVec
        .
        toRows
        where
            classifySingleVec :: Vector -> Label
            classifySingleVec x =
                if
                    mahalanobis min1 covariance1 x > mahalanobis min2 covariance2 x
                then
                    labelNeg
                else
                    labelPos
```

Die Testdaten werden als Matrix übergeben, deren Zeilen die Features der zu klassifizierenden Samples enthalten. Für jedes Sample wird jeweils berechnet, welche Klasse gemessen mit der Mahalanobis-Distanz näher liegt und dementsprechend sortiert.

1.3 Kompilieren des Programms

1.3.1 Abhängigkeiten

- git (siehe https://git-scm.com/)
- stack (siehe https://docs.haskellstack.org/)

1.3.2 Kompilieren

```
$ git clone https://github.com/EsGeh/pattern-recognition
$ git checkout exercise3-release
$ stack setup
$ stack build
```

1.3.3 Ausführen mittels Stack

```
$ stack exec patternRecogn-exe
```