Mustererkennung - Übung6

Semjon Kerner, Philip Schmidt, Samuel Gfrörer

2016-12-2

1 Fischerdiskrimination

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
|   |   `-- Perceptron.hs
|   `-- ...
```

Das Programm teilt sich auf in eine ausführbare Datei (siehe Verzeichnis "./app") und eine Bibliothek (siehe Verzeichnis "./src"). Für diese Übung relevanter Quellcode:

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

- "./src/PatternRecogn/Perceptron.hs": Hier befindet sich die eigentliche Funktionalität des Klassifizierungsalgorithmus

1. Eingabe- und Ausgabedateien

Im Ordner "./resource" befinden sich die Trainingsdatensätze und der Testdatensatz. Die Ausgabe erfolgt über die Standardausgabe.

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":

"./app/Main.hs":

```
42 main :: IO ()
43 main =
44     handleErrors $
45     do
46         mapM_
47             (uncurry4 testWithData . uncurry testParamsFromLabels) $
48             allPairs [3,5,7,8]
49     where
50         handleErrors x =
51             ...
```

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

"./app/Main.hs":

```
66 testWithData :: FilePath -> FilePath -> Label -> Label -> ErrT IO ()
67 testWithData trainingFile1 trainingFile2 label1 label2 =
68     do
69         ...
70         testInput <-
71             readTestInput
72                 trainingFile1 trainingFile2
73                 label1 label2
74             :: ErrT IO (AlgorithmInput, Vector)
75         testPerceptron label1 label2 testInput
76             >>= \quality -> liftIO $ putStrLn $ concat $ ["perceptron quality:", show $ qual
```

Hier werden die geladenen Daten mittels Perzeptron-Lernen klassifiziert und deren Trefferquote gemessen.

Das Durchlauf beim Lernen funktioniert so: Es werden *alle* Trainingsdaten durchlaufen (zunächst die mit Label1, dann die mit Label2).

"./src/PatternRecogn/Perceptron.hs":

```haskell
perceptronStepAll set1 set2 param =
    perceptronStep (-1) set1
    .
    perceptronStep 1 set2
    $
    param
```

Dabei wird für jedes Sample der Trainingsmenge der Gewichtsvektor β korrigiert, falls er das Sample nicht richtig klassifiziert:

"./src/PatternRecogn/Perceptron.hs":

```haskell
perceptronStep :: Label -> Matrix -> ClassificationParam -> ClassificationParam
perceptronStep expectedLabel set param =
    foldl conc param $ Lina.toRows set
    where
        conc :: ClassificationParam -> Vector -> ClassificationParam
        conc beta y =
            let estimatedClass = classifySingleSample_extended (-1, 1) beta y
            in
                if estimatedClass == expectedLabel
                then beta
                else
                    if estimatedClass < 0
                    then beta + y
                    else beta - y
```

Dieses Verfahren wird (bis maximal 1000fach) iteriert, so lange bis der Fehler klein genug ist. Trotzdem bei jeder Iteration ("perceptronStepAll") alle Trainingsdaten durchlaufen werden ist die Laufzeit bemerkenswert schnell.

1.2.3 Ergebnis

Dies ist die Ausgabe des Programms:

```
$ ./scripts/run.sh
-----------------------------------------------
classifying to labels [3,5] in files ["resource/train.3","resource/train.5"]
perceptron quality:0.911042944785276
-----------------------------------------------
classifying to labels [3,7] in files ["resource/train.3","resource/train.7"]
perceptron quality:0.9776357827476039
-----------------------------------------------
classifying to labels [3,8] in files ["resource/train.3","resource/train.8"]
perceptron quality:0.9668674698795181
-----------------------------------------------
classifying to labels [5,7] in files ["resource/train.5","resource/train.7"]
```

```
perceptron quality:0.9869706840390879
-----------------------------------------------
classifying to labels [5,8] in files ["resource/train.5","resource/train.8"]
perceptron quality:0.9478527607361963
-----------------------------------------------
classifying to labels [7,8] in files ["resource/train.7","resource/train.8"]
perceptron quality:0.9840255591054313
```

Die Klassifizierungsqualität schwankt deutlich, ist aber in allen Fällen über 90%.

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 exercise6-release
$ stack setup
$ stack build
```

1.3.3 Ausführen mittels Stack

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