Mustererkennung WiSe 12/13
Übung 1

Lutz Freitag, Sebastian Kürten

Aufgabe 1 - Visualisierung der Pendigts Eingabedaten

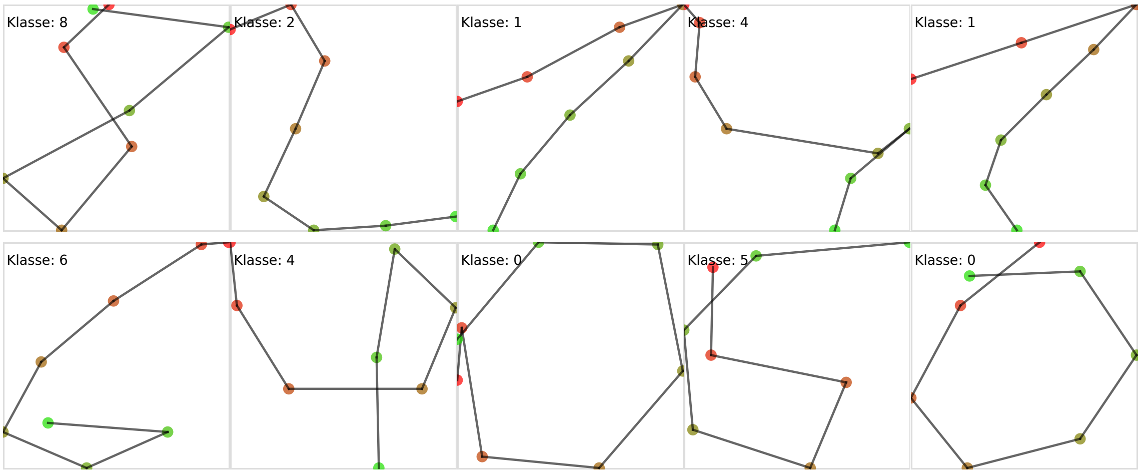

Figure 1: Darstellung der ersten 10 Trainingsdaten

- Aufruf zum Plotten der ersten 10 Eingabedaten:

  ```
  ./ub1.a1.b.sh
  ```

  bzw.:

  ```
  java -cp bin:lib/* ub1.a1.TestDrawImages
          data/pendigits-training.txt output/a1/pen 10
  ```

- Abbildung 1 zeigt die Ausgabe
- Die Klasse `pendigits.PendigitReader` implementiert statische Methoden für das Einlesen der Eingabedaten. Diese liefern eine `Collection` mit Elementen vom Typ `pendigits.PendigitRecord`.

Aufgabe 2 - 1-Nearest-Neighbours

- Aufruf zum Klassifizieren aller Testdaten und zum Berechnen der Trefferquote:

  ```
  ./ub1.a2.sh
  ```

  bzw.:

  ```
  java -cp bin:lib/* ub1.a2.TestPendigitDetection
          data/pendigits-training.txt data/pendigits-testing.txt
  ```

- Ausgabe:

```
#records training: 7494
#records testing: 3498
success rate: 97.74%
failure rate: 2.26%
```

- Die abstrakte, generische Klasse `classificator.Classificator` implementiert Methoden zur Klassifizierung von Eingabedaten. Diese wird vom `ub1.a2.PendigitClassificator` erweitert und implementiert die Abstandsfunktion, die vom Klassifizierungsalgorithmus benötigt wird.

Wichtigste Codestellen

Main-Methode

```
public static void main(String[] args) throws IOException {
        if (args.length != 2) {
                System.out.println(”TestPendigitDetection
                        <training data> <testing data>”);
                System.exit(1);
        }

        String filePathTraining = args[0];
        String filePathTesting = args[1];

        List<PendigitRecord> recordsTraining = PendigitReader
                        .readAtFilePath(filePathTraining);
        List<PendigitRecord> recordsTesting = PendigitReader
                        .readAtFilePath(filePathTesting);

        System.out.println(String.format(”#records training: %d”,
                        recordsTraining.size()));
        System.out.println(String.format(”#records testing: %d”,
                        recordsTesting.size()));

        PendigitClassificator classificator = new PendigitClassificator();

        int successCount = 0;
        int n = recordsTesting.size();
        for (PendigitRecord record : recordsTesting) {
                int trueClass = record.getClassNumber();
                int guessedClass = classificator.classify(record, recordsTraining, 1);
                boolean success = trueClass == guessedClass;
                if (success) {
                        successCount += 1;
                }
        }
        double successRate = successCount / (double) n;
        double failureRate = 1 - successRate;
        System.out.println(String.format(”success rate: %.2f%%”,
                        successRate * 100));
        System.out.println(String.format(”failure rate: %.2f%%”,
                        failureRate * 100));
}
```

Methoden zur Bestimmung des nächsten Nachbarn

```
public int classify(T record,
                Collection<T> recordsTraining, int k)
{
        // first compute the distances to all training records.
        List<WeightedRecord<T>> neighbors = calculateNeighborDistances(record,
                        recordsTraining);
        // than classify according to the k nearest neighbors.
        return classify(record, neighbors, k);
}

private List<WeightedRecord<T>> calculateNeighborDistances(
                T record,
                Collection<T> recordsTraining)
{
        // create the list to store records along with distances
        List<WeightedRecord<T>> list = new ArrayList<WeightedRecord<T>>();
        for (T other : recordsTraining) {
                // compute distance
                double dist = distance(record, other);
                // and store the record with this distance
                WeightedRecord<T> weightedRecord = new WeightedRecord<T>(other,
                                dist);
                list.add(weightedRecord);
        }
        // sort the list with ascending distance
        Collections.sort(list);
        return list;
}

public int classify(T record,
                List<WeightedRecord<T>> neighbors, int k)
{
        // create a list of candidates with duplicates
        List<Integer> classCandidates = new ArrayList<Integer>();
        // for the k nearest neighbors
        for (int i = 0; i < k; i++) {
                // find the i'th nearest neighbor
                WeightedRecord<T> wr = neighbors.get(i);
                // determine its class number
                int classNumber = wr.getRecord().getClassNumber();
                // and add it to the candidate list
                classCandidates.add(classNumber);
        }
        // from the candidates, pick the one that occurs most often
        int classNumber = pickClass(classCandidates);
        return classNumber;
}

public double distance(PendigitRecord record, PendigitRecord other)
{
        double sum = 0;
        for (int i = 0; i < record.getNumberOfCoordinates(); i++) {
                Coordinate ca = record.getCoordinate(i);
                Coordinate cb = other.getCoordinate(i);
                sum += Math.pow(cb.getX() - ca.getX(), 2);
                sum += Math.pow(cb.getY() - ca.getY(), 2);
        }
        return Math.sqrt(sum);
}
```