Prof. R. Rojas

Mustererkennung, WS17/18

Übungsblatt 1

Boyan Hristov, Matrikelnummer: 4980301

25. Oktober 2017

Link zum Git Repository: https://github.com/BoyanH/Freie-Universitaet-Berlin/tree/master/MachineLearning/Homework1

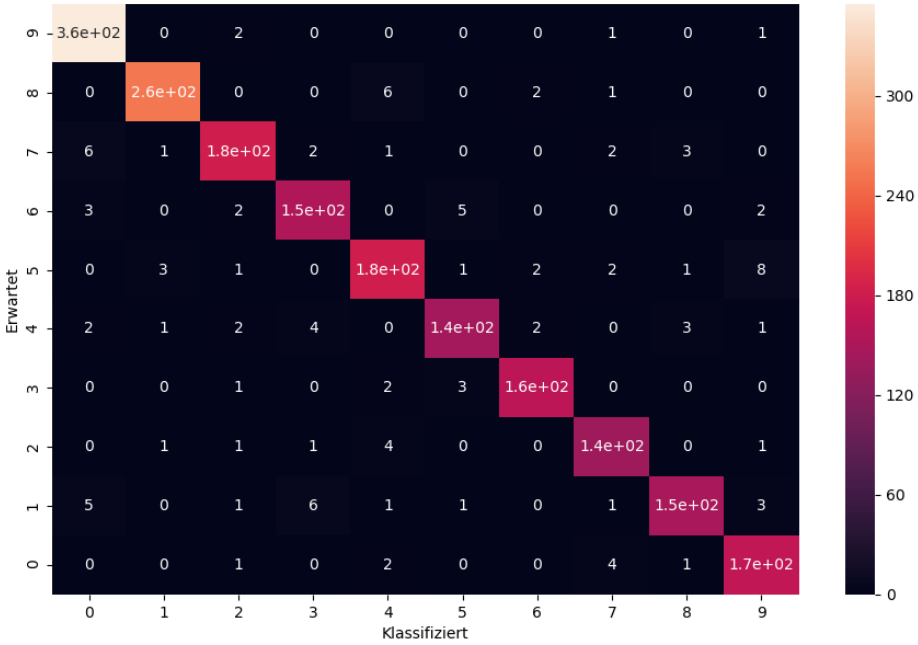
Beschreibung

Ziffererkennung mit kNN-Classifier. Dabei wurde eine simple lineare kNN Klassifizierung ohne kd-Baum benutzt. Die Eingabedaten (Grayscale Pixel von Bilder) wurden als Vektoren in n-Dimensionalen Raum (n=256) betrachtet, ohne weitere Verarbeitung von dem Merkmalen.

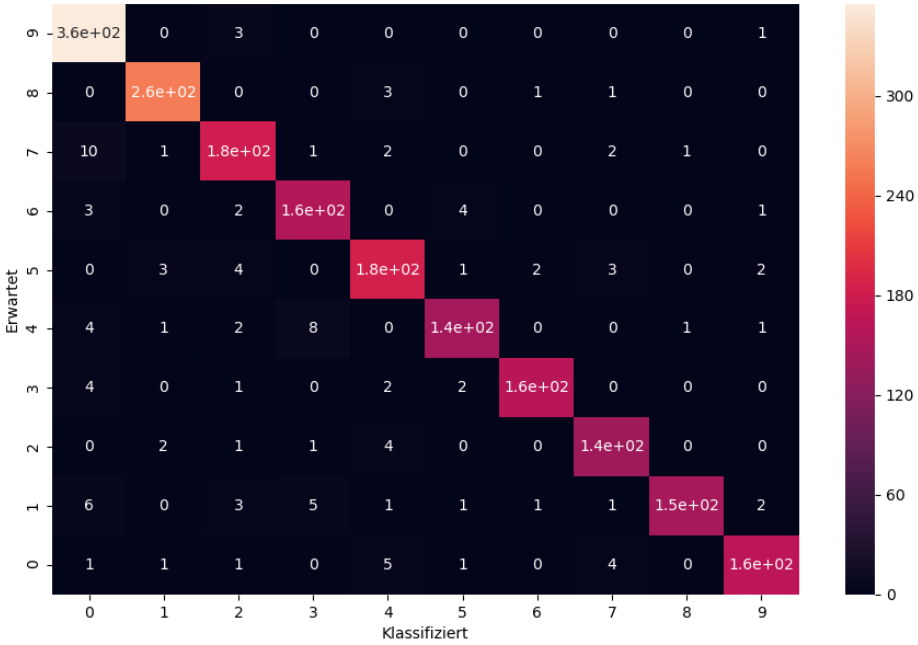
Fehlerrate

Das ist die Ausgabe des Programs und damit auch die Fehlerrate

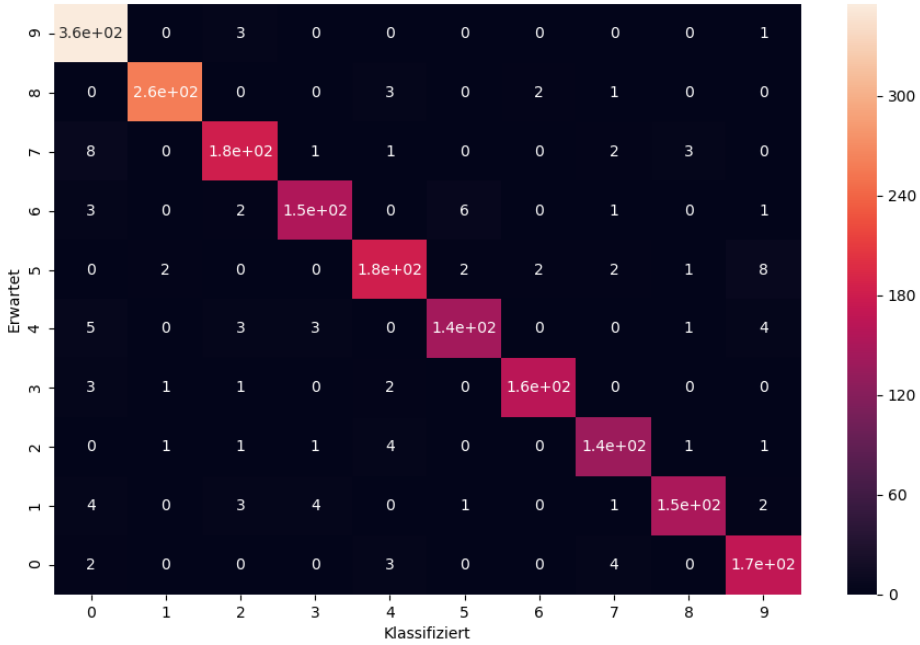
```
Error rate for k=1 is 5.630293971101146%
Error rate for k=2 is 5.879422022919781%
Error rate for k=3 is 5.5306427503736915%
```


Erwartet
Klassifiziert
9 3.6e+02 0 2 0 0 0 0 1 0 1
8 0 2.6e+02 0 0 6 0 2 1 0 0
7 6 1 1.8e+02 2 1 0 0 2 3 0
6 3 0 2 1.5e+02 0 5 0 0 0 2
5 0 3 1 0 1.8e+02 1 2 2 1 8
4 2 1 2 4 0 1.4e+02 2 0 3 1
3 0 0 1 0 2 3 1.6e+02 0 0 0
2 0 1 1 1 4 0 0 1.4e+02 0 1
1 5 0 1 6 1 1 0 1 1.5e+02 3
0 0 0 1 0 2 0 0 4 1 1.7e+02
0 1 2 3 4 5 6 7 8 9
300 240 180 120 60 0

k=1


Erwartet
Klassifiziert
9 3.6e+02 0 3 0 0 0 0 0 0 1
8 0 2.6e+02 0 0 3 0 1 1 0 0
7 10 1 1.8e+02 1 2 0 0 2 1 0
6 3 0 2 1.6e+02 0 4 0 0 0 1
5 0 3 4 0 1.8e+02 1 2 3 0 2
4 4 1 2 8 0 1.4e+02 0 0 1 1
3 4 0 1 0 2 2 1.6e+02 0 0 0
2 0 2 1 1 4 0 0 1.4e+02 0 0
1 6 0 3 5 1 1 1 1 1.5e+02 2
0 1 1 1 0 5 1 0 4 0 1.6e+02
0 1 2 3 4 5 6 7 8 9
300 240 180 120 60 0

k=2



k=3

Code

```
import os
import math

import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt

dir_path = os.path.dirname(os.path.realpath(__file__))
trainingSetData = []

def extractDataFromLine(line):
 line = line.replace(' \n', '') # clear final space and new line chars

 return list(map(float, line.split(' '))); # map line to a list of floats

def train():
 explicitPathRead = os.path.join(dir_path, './Dataset/train')
 f = open(explicitPathRead, 'r')

 for line in f:
  line = line.replace(' \n', '') # clear final space and new line chars
  currentDigitData = extractDataFromLine(line)
  trainingSetData.append(currentDigitData)

 # TODO: maybe construct a kd-tree out of data

def test(knnCount):
 explicitPathRead = os.path.join(dir_path, './Dataset/test')
 f = open(explicitPathRead, 'r')

 errors = 0
 testsCount = 0
```

```
 confusionsMatrix = [[0 for x in range(10)] for y in range(10)] # 10 by 10 matrix

 for line in f:
  currentLineData = extractDataFromLine(line)
  result = classify(currentLineData, knnCount)
  expected = str(int(currentLineData[0]))

  confusionsMatrix[int(expected)][int(result)] += 1

  testsCount += 1
  if result != expected:
   errors += 1

 errorRate = errors / testsCount
 printConfusionsMatrix(confusionsMatrix, knnCount)
 print("Error rate for k={0} is {1}%".format(knnCount, errorRate * 100))

def classify(digitData, knnCount):
 kNN = getKNN(trainingSetData, digitData, knnCount)
 labelsOfKNN = list(map(getLabel, kNN)) # intentionally as integers, to make mapping easier
 digitRepetitions = [0 for x in range(10)]

 for label in labelsOfKNN:
  digitRepetitions[label] += 1

 return str(digitRepetitions.index(max(digitRepetitions)))


def getKNN(trainingSetData, digitData, k):
 knn = [] # distance to the kNN
 knnToDataMapper = {}

 for trainingDigitData in trainingSetData:
  currentDistance = getDistanceBetweenPoints(getCoords(digitData), getCoords(
    trainingDigitData))

  if len(knn) < k or max(knn) > currentDistance:
   knn.append(currentDistance)
   knn.sort()
   knnToDataMapper[currentDistance] = trainingDigitData
   knn = knn[:k]

 return list(map(lambda x: knnToDataMapper[x], knn))


def getCoords(data):
 return data[1:]

def getLabel(data):
 return int(data[0])

def getDistanceBetweenPoints(a, b):
 if len(a) != len(b):
  raise Exception('Points must be in the same n-dimensional space to calculate distance!')

 squares = 0

 for i in range(len(a)):
  squares += math.pow(a[i] - b[i], 2)

 return math.sqrt(squares)

def printConfusionsMatrix(matrix, k):
 explicitImgPath = os.path.join(dir_path, './Plots/confusion_matrix_for_k_{0}.png'.format(k
   ))
 digits = [str(x) for x in range(10)];

 df_cm = pd.DataFrame(matrix, index = digits,
```

```
 columns = digits.reverse() )

 plt.figure(figsize = (11,7))
 heatmap = sn.heatmap(df_cm, annot=True)

 heatmap.set(xlabel='Klassifiziert', ylabel='Erwartet')

 plt.savefig(explicitImgPath, format='png')

train()
test(1)
test(2)
test(3)
```