Fernfachhochschule Schweiz
Zürich | Basel | Bern | Brig

Mitglied der SUPSI

3. Semesterarbeit

Handschrifterkennung mit Scikit-learn

EDS, Einführung in Data Science, INF-P-IN010, BE2, HS20/21, Dr. Paoli Beatrice

Inhaltsverzeichnis

1. Abstract

In dieser Arbeit wird sich damit befasst, ein einfaches neuronales Netzwerk zu erstellen. Das Ziel ist es, die eigene Handschrift durch ein neuronales Netzwerk erkennen lassen zu können. Die nötigen und optimalsten Einstellungen werden gesucht für dieses Beispiel. Das Erstellen eines eigenen Datensatzes wird thematisiert und schlussendlich durch das trainierte Netzwerk geladen. Der Versuch ist geglückt, jedoch scheint das Resultat nicht vollends zufriedenstellend zu sein. Es bleibt die Frage ob dies das richtige Format für den gewählten Versuch ist.

2. Einleitung

Das Training eines neuronalen Netzwerkes wird grob betrachtet und an einem Beispiel dargestellt. Daraus wird versucht eine optimale Einstellung für unser Beispiel zu finden. Mit dem so eingestellten neuronalen Netzwerk, wird anschliessend versucht die eigene Handschrift einzulesen und erkennen zu lassen. Das Bereitstellen der eigenen Daten wird aufgezeigt.

3. Material

Zur Verwendung kamen die IDE PyCharm [1] und Programmiersprache Python [2]. Das Framework und Training Datasets wurde von Scikit-learn [3] verwendet. Ein genauerer Blick auf die Datasets gibt es im Machine Learning Repository [4]. Grundsätzliches Vorgehen wurde im Buch, Mining the Social Web [5] aus dem Kapitel 3 entnommen. Zur Bildbearbeitung wurde Gimp[6] verwendet.

Scikit-learn in ihren eigenen Worten:

```
sklearn is a Python module integrating classical machine
learning algorithms in the tightly-knit world of scientific Python
packages (numpy, scipy, matplotlib).

It aims to provide simple and efficient solutions to learning
problems
that are accessible to everybody and reusable in various contexts:
machine-learning as a versatile tool for science and engineering.
```

Vorangehend sollten folgende Pakete installiert werden:

```
pip3 install scikit-learn
pip3 install numpy
pip3 install matplotlib
```

4. Methoden

Zuerst wird ein Dataset benötigt, um das neuronale Netzwerk trainieren zu können. Dieses selbst herzustellen wäre ein enormer Aufwand, daher bietet es sich an, auf vorhandene Sammlungen zurück zu greifen.

```
digits = datasets.load_digits()
```

Um das neuronale Netzwerk zu testen werden aus dem Dataset zwei verschiedene Teile gemacht. Zum einen ein Test-Teil und zum anderen ein Train-Teil. Die Datensätze in Test bekommt das Netzwerk erst zu Gesicht nach dem Training zur Verifizierung. Mit Train wird das neuronale Netzwerk trainiert und auf seine Aufgabe mit bezeichneten Beispielen eingestimmt.

```
train_test_split(X, y, test_size=0.25, random_state=42)
```

Im Beispiel wird ¼ der Daten in den Test-Teil gesplittet. Damit bei wiederholten Versuchen nicht immer dieselben Elemente im Test-Teil sind wird noch ein Zufallswert hinzugefügt. Nach diesem Wert werden die Daten vor dem Split gemischt.

Das Training wird mit dem MLPClassifier konfiguriert. Wie gross der Hidden Layer sein soll, wie viele Iterationen maximal gemacht werden sollen, wann das Training abgebrochen werden soll weil es im Verlauf zu wenig Verbesserung mehr gibt.

```
MLPClassifier(hidden_layer_sizes=(120,), max_iter=100, alpha=1e-4,
              solver='adam', verbose=10, tol=1e-4, random_state=1,
              learning_rate_init=.1)
```

Um genauer zu erfahren wie dieser essenzielle Teil funktioniert, wird dringend empfohlen die Dokumentation [7] dazu zu lesen.

Die Durchführung des Trainings wird mit dem dafür vorgesehenen Dataset X_train erstellt. Das Dataset y enthält die entsprechenden Labels. Nach dem Training wird ein trainiertes MLP-Modell zurückgegeben.

```
fit(X_train, y_train)
```

Die Daten, die es zu prüfen gilt, werden anschliessend durch das trainierte MLP-Modell gespielt und die vermeintlich erkannten Labels zurückgegeben.

```
predict(X_test)
```

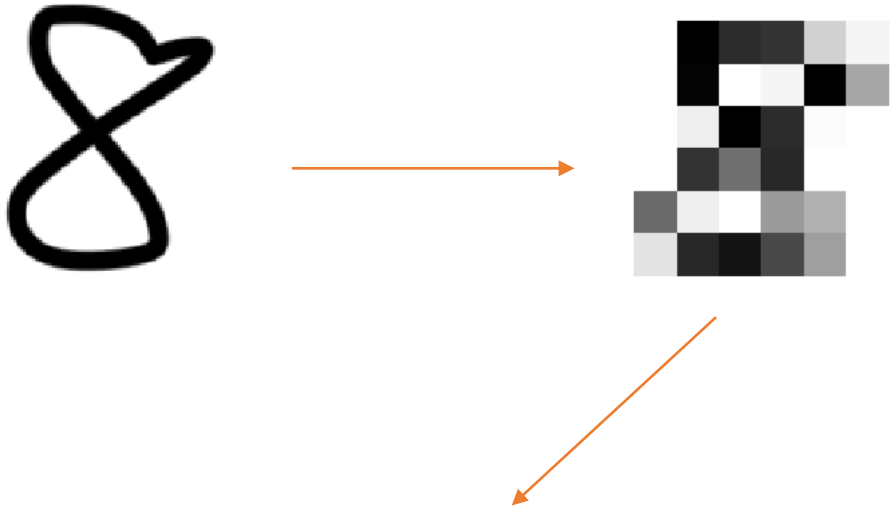
Eigene Daten bereitstellen:

```
calcArray = np.ones(64)

np.subtract(calcArray, np.reshape(mpimg.imread("Handschrift/" +
file)[..., :1], 64))
```

Um ein eigenes Bild bereit zu stellen, wurde von matplotlib die Funktion imread verwendet. Hier gab es mehrere Probleme, die gelöst werden mussten. Die Bilder wurden mit Gimp komprimiert zu einer Grösse von 8x8 Pixel. Diese Datei wird als multidimensionales numpy.array ingelesen. Um dieses Array verwenden zu können, muss es in ein richtiges Format gebracht werden. Mit reshape von numpy kann das Format eines Arrays angepasst werden. Die Bilder werden in den Inversen Werten, die für unseren Test benötigt werden, eingelesen. Mit der Subtraktion durch ein separat erstelltes Einserarray wird dem abgeholfen.

Beispiel für die Zahl 8:



```
[0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       1.        , 0.82745098, 0.8       , 0.18039215, 0.03921568,
       0.        , 0.        , 0.        , 0.98431372, 0.        ,
       0.03921568, 1.        , 0.34901959, 0.        , 0.        ,
       0.        , 0.06666666, 1.        , 0.82745098, 0.01176471,
       0.        , 0.        , 0.        , 0.        , 0.8       ,
       0.56078431, 0.84313725, 0.        , 0.        , 0.        ,
       0.        , 0.58823529, 0.06666666, 0.        , 0.39215684,
       0.30588233, 0.        , 0.        , 0.        , 0.10980392,
       0.84313725, 0.92549019, 0.71764705, 0.37647057, 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        ]
```

5. Resultate

Das neuronale Netzwerk kann trainiert und mit eigenen Daten zugeführt werden. Es wurde keine hundertprozentige Erkennung der eigenen Daten erreicht.

Im MLPClassifier können Einstellungen vorgenommen werden, die auf das Training des neuronalen Netzwerkes einwirken. Folgende Parameter wurden betrachtet.

```
hidden_layer_sizes=hidden_layer_sizes,
The ith element represents the number of neurons in the ith
hidden layer.

max_iter=max_iter,
Maximum number of iterations. The solver iterates until convergence
(determined by 'tol') or this number of iterations. For stochastic
solvers ('sgd', 'adam'), note that this determines the number of
epochs
(how many times each data point will be used), not the number of
gradient steps.

tol=tol,
Tolerance for the optimization. When the loss or score is not
improving
by at least ``tol`` for ``n_iter_no_change`` consecutive iterations,
unless ``learning_rate`` is set to 'adaptive', convergence is
considered to be reached and training stops.

learning_rate_init : double, default=0.001
    The initial learning rate used. It controls the step-size
    in updating the weights. Only used when solver='sgd' or 'adam'.
```

Die beste Einstellung für das verwendete Beispiel wurde gefunden durch die Kontrolle der benötigten Iterationen bis zum Stoppen des Trainings. Das Kriterium für das Beenden soll nicht durch die maximale Iterationszahl erreicht werden, sondern durch zu kleine Verbesserungen in 10 aufeinanderfolgenden Iterationen.

```
MLPClassifier(hidden_layer_sizes=(120,), max_iter=100, alpha=1e-4,
                  solver='adam', verbose=10, tol=1e-4, random_state=1,
                  learning_rate_init=.1)
```

Es kann nicht abschliessend eine einzelne Einstellung für jedes neuronale Netzwerk erstellt werden. Die Parameter müssen für verschiedene Datensätze angepasst werden.

Die Beobachtungen haben gezeigt, dass die Erhöhung der Iterationen oder der Anzahl von Neuronen nicht mit einer Erhöhung der Genauigkeit einher gehen.

6. Diskussion

Gefahr von overfitting – underfitting
Eine grosse Schwierigkeit wird in der Erkennung des wahren Ergebnisses gesehen. Zu beurteilen ob das neuronale Netzwerk nicht overfitted oder underfitted ist, erscheint als Essenziel. Hier sollte vor der Auswertung aller Daten sichergestellt werden, dass mit einem vertrauenswürdigen Netzwerk gearbeitet wird. Ein Overfit beschreibt ein Netzwerk, dass zu stark auf die Trainingsdaten eingespielt ist. Ein Underfit wurde zu wenig genau trainiert auf seine Aufgabe.

Schlechte Qualität der bereitgestellten Daten
In dieser Aufgabe ging es schlussendlich darum die eigene Handschrift erkennen zu lassen. Die erstellten Bilder von 8x8 Pixel lassen nur noch grob Merkmale einer spezifischen Handschrift erkennen. Für einen weiteren Versuch um dieses Thema muss die Pixelzahl erhöht werden, um feinere Details erkennen zu lassen.

Es stellt sich die generelle Frage:
Sind 8x8 Pixel noch «Handschrift»?

Eigene Daten herzustellen benötigt sehr viel Zeit. Um dies effizienter zu gestalten, sollte bei grösseren Projekten in eine Automatisierung in Betracht gezogen werden.

7. Quellen

[1]	PyCharm: https://www.jetbrains.com/de-de/pycharm/
[2]	Python: https://www.python.org
[3]	Scikit-learn: https://scikit-learn.org/stable/
[4]	Datasets: https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits
[5]	Mining the Social Web: https://www.orellfuessli.ch/shop/home/artikeldetails/ID64546124.html?ProvID=10917735&&ProvID=10917735&gclid=CjwKCAjw0On8BRAgEiwAincsHOg_dd9lIxqD1U0FYIS6bnCVg3MkCEby7icShFhO6VRw6jW5QT9Z2RoCclgQAvD_BwE&gclsrc=aw.ds
[6]	Gimp: https://www.gimp.org
[7]	MLPClassifier: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html?highlight=mlpclassifier#sklearn.neural_network.MLPClassifier

8. Anhang

```
from sklearn import datasets
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
import numpy as np
import os
from matplotlib import image as mpimg

digits = datasets.load_digits()

# Rescale the data and split into training and test sets
X, y = digits.data / 255., digits.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
random_state=42)

# Einstellungen neuronales Netzwerk
mlp = MLPClassifier(hidden_layer_sizes=(120,), max_iter=100, alpha=1e-4,
                    solver='adam', verbose=10, tol=1e-4, random_state=1,
                    learning_rate_init=.1)

# Training neuronales Netzwerk
mlp.fit(X_train, y_train)

print()
print("Training set score: {0}".format(mlp.score(X_train, y_train)))
print("Test set score: {0}".format(mlp.score(X_test, y_test)))
print()
print("---------")
print("---------")

# Input des Testsets
predicted = mlp.predict(X_test)
print("Vergleich mit dem Testset:")
print("---------")
for i in range(10):
    print('Ground Truth: {0}'.format(y_test[i]))
    print('Predicted: {0}'.format(predicted[i]))
print("---------")
print("---------")

calcArray = np.ones(64)
imgArray = []
imgOrder = []

for file in os.listdir("Handschrift"):
    if file.endswith('.png'):
        imgOrder.append(file)
        image = np.subtract(calcArray, np.reshape(mpimg.imread("Handschrift/" +
file)[..., :1], 64))
        imgArray.append(image)

# Input des eigenen Datensets
predicted = mlp.predict(imgArray)

count = 0
print("Vergleich mit den eigenen Daten:")
print("---------")
for pred in predicted:
    print("Test - " + str(count+1))
    print("Input Number:        " + str(imgOrder[count]))
    print("Predicted Number:    " + str(predicted[count]))
    count = count + 1
    print("--------")
```