“Klassen und Ableitungen”

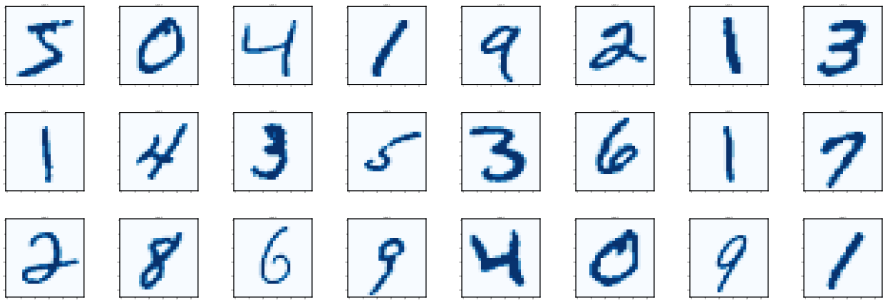
Übungsaufgabe 1

Informatik 2 für Biomedical Engineering

Technische Universität Graz

Abstract

Die MNIST Database of handwritten digits[1] ist ein weit verbreiteter Datensatz für Klassifizierungsaufgaben. Er besteht aus 65.000 Samples handgeschriebener Buchstaben. Im Rahmen dieses Assignments ist es Ihre Aufgabe, ein Package zu implementieren, welches den Datensatz lesen und für die nächsten Schritte vorbereiten kann.



---

[1] http://yann.lecun.com/exdb/mnist/

1. Tasks

Im Rahmen dieses Assignments schreiben Sie ein Package bestehend aus 3 Klassen. Einer abstrakten Basisklasse und zwei Ableitungen. Die Datei `assignment_1.py` dient lediglich für Sie zum Testen Ihrer eigenen Abgabe. Nutzen Sie diese! Bei allen Tests wird diese Datei ersetzt.

1.1. Abstrakte Reader Klasse (4pt)

Importieren Sie das Hilfspackage für abstrake Klassen. Sie können dafür folgenden Code verwenden:

```
from abc import ABC, abstractmethod
```

Erstellen Sie eine Klasse namens `'Reader'`. Diese soll von der `ABC` Klasse abgeleitet sein.

1.1.1. Funktionen

`__init__(self, data_path):` Der Konstruktor der Klasse soll einen Parameter haben, welcher den Pfad zu den Daten definiert. Überprüfen Sie bereits hier, ob der Pfad existiert. Sollte dies nicht der Fall sein, verwenden Sie einen `FileNotFoundError`.

`(abstract)read_data(self):` Die abstrakte Methode `read_data` wird von Ihren Erben implementiert. In der abstrakten Klasse reicht daher folgender Code im Körper der Funktion:

```
raise NotImplementedError
```

Verwenden Sie den Decorator `@abstractmethod` um dies eindeutig als abstrakte Methode zu definieren. (`abstractmethod` muss aus dem `abc` package importiert werden!)

`dump(self, dump_path)` Die dump Funktion soll die geladenen Daten als pickle speichern. Als Parameter wird der vollständige Pfad (Ordner + Dateiname) übergeben. Überprüfen Sie, ob der Pfad vorhanden ist. Erstellen Sie ein Objekt mit folgender Struktur und speichern Sie es als pickle:

```
dump_data = {
            "X": None, # Die Features der Samples
            "y": None, # Die Labels der Samples
            "num_rows": None, # Anzahl der Pixelreihen pro Sample
            "num_cols": None, # Anzahl der Pixelspalten pro Sample
            } #Ersetzen Sie die None im Beispiel mit Ihren Werten!
```

`plot_sample(self, index, store_path)` : Diese Funktion dient zum Visualisieren eines einzelnen Samples. Verwenden Sie Funktion [imshow] von Matplotlib[2]. Diese benötigt eine Liste aus Listen (numpy listen Matrix). Zum Erstellen dieser Matrix können Sie `np.split`[3] oder `np.reshape`[4] verwenden. **Hinweis:** Verwenden Sie diese Funktion zum Testen, ob Ihr Reader die Daten korrekt geladen hat.

1.1.2. Properties

`X:` Diese Variable beinhaltet die Datenpunkte (Features) für alle Samples. Es ist eine Liste aus Listen. Beispiel dieser Struktur für die Samples `s01` und `s02` welche jeweils 4 Datenpunkte(Features) besitzen.

```
[[s01_1, s01_2, s01_3, s01_4], [s02_1, s02_2, s02_3, s02_4]]
```

Das Schreiben der Variable von außen soll über den Setter blockiert werden.

`y:` Diese Variable beinhaltet die Labels für alle Samples. Es handelt sich hier um eine einfache Liste. Jedes Sample hat nur 1 Label (eben um welche Ziffer es sich handelt).

`num_rows:` Die Anzahl der Datenreihen pro Sample. Da jedes Sample ein Bild ist, entspricht dies der vertikalen Auflösung.

`num_cols:` Die Anzahl der Datenspalten pro Sample. Da jedes Sample ein Bild ist, entspricht dies der horizontalen Auflösung.

1.2. MNIST Reader (5pt)

Diese Klasse hat die Aufgabe, die originalen MNIST Daten zu laden. Überschreiben Sie die `read_data()` Methode der Mutterklasse. Als Parameter nimmt diese Funktion 2 Tuples entgegen. Einmal für die Features-Datei und einmal für die Labels. Jedes dieser Tuples besteht aus den Dateinamen und der dazugehörigen *Magic Number*. Folgendes ist ein Beispiel-Aufruf:

[2]https://matplotlib.org/api/_as_gen/matplotlib.pyplot.imshow.html
[3]https://docs.scipy.org/doc/numpy/reference/generated/numpy.split.html
[4]https://docs.scipy.org/doc/numpy/reference/generated/numpy.reshape.html

```
mnist_reader.read_data(features=("train-images-idx3-ubyte.gz", 2051),
                       labels=("train-labels-idx1-ubyte.gz", 2049))
```

Überprüfen Sie im ersten Schritt, ob beide Dateien vorhanden sind. Zum öffen und Lesen der Dateien, verwenden Sie das `gzip`[5] package. Mit `gzip.open()` verhalten sich geöffnete Zip Archive für Sie weitestgehend wie normale Dateien.

1.2.1. Lesen der Daten

Als zweites müssen Sie die Magic Number verifizieren. Lesen Sie hierzu die ersten 4 Byte der jeweiligen Datei und vergleichen Sie den Integer-Wert. Sollte Sie nicht übereinstimmen, informieren Sie den Nutzer darüber mit einer entsprechenden Exception. Sie können zum Lesen der Datei struct.unpack[6] verwenden. Als Flag verwenden Sie hier `>i`. (`>` beschreibt die Endianness, bzw die Position des most significant Bit)
Der letzte Check ist der Vergleich der sample Anzahl in den beiden Dateien. Öffnen Sie beide Dateien und springen Sie an die Position 4 (mögliche Wege: `read(4)`, oder `seek(4)`). Lesen Sie das die nächsten 4 Byte (wieder mit `>i`) und und vergleichen Sie die Werte.
Laden Sie als nächstes die Labels. Öffnen Sie die Datei und springen Sie an Position 8 . Lesen Sie danach die restliche Datei mit der `struct.unpack` Flag `'B'` (für unsigned Char mit der Größe von jeweils 2 Byte). `struct.iter_unpack()`[7] ist hilfreich hierfür. Die gelesenen Labels speichern Sie in der Property `y`.
Widmen Sie sich nun den Features. Öffnen Sie die entsprechende Datei und springen Sie an Position 8. Die nächsten beiden Integer sind die Anzahl der Reihen gefolgt von der Anzahl an Spalten. Verwenden Sie auch hier die Flag für Integers (`'>i'`) diese Werte zu lesen und speichern Sie sie in die passenden Properties. Berechnen Sie nun die Anzahl der Datenpunkte pro Sample. Lesen Sie den Rest der Datei ein. Für jedes Sample lesen Sie die Anzahl der Datenpunkte ein. Diese sind jeweils wieder unsigned Chars (Flag: `'B'`. Speichern Sie diese Liste aus Listen in die Property `X`.
Hinweis: Verwenden Sie private Funktionen (`_` - Prefix). Zum Beispiel: Prüfen der Magic Number oder Extrahieren der Labels und Features.

[5] https://docs.python.org/3/library/gzip.html
[6] https://docs.python.org/3.5/library/struct.html#struct.unpack
[7] https://docs.python.org/3.5/library/struct.html#struct.iter_unpack

1.3. Pickle Reader (3pt)

Erstellen Sie eine zweite abgeleitete Reader Klasse. Die Aufgabe der Klasse `PickleReader` ist es, pickle Dateien (die über die `dump()` Methode gespeichert wurden) zu lesen. Überschreiben Sie dafür die `read_data` Methode der Mutterklasse. Der Parameter der Funktion ist der Pfad zum Pickle. Dabei sollten die Member Variablen/Properties `X`, `y`, `num_rows`, `num_cols` korrekt gelesen und wiederhergestellt werden. Überprüfen Sie auch, ob die gewünschte (pickle)Datei existiert.
Hinweis: Vergessen Sie in den beiden abgeleiteten Klassen nicht, den Konstruktor der Mutterklasse (via `super().__init__()`) aufzurufen. Sonst stehen Ihnen die dort implementierten Funktionen (zum Speichern und Plotten) nicht zur Verfügung.

1.4. Package (3pt)

Implementieren Sie Ihre Abgabe als eigenständiges python Package. Mit folgendem Code soll man Ihr dies importieren können. Erstellen Sie hierfür die `__init__.py` Datei in importieren Sie dort ihre Klassen.
Hinweis: verwenden Sie lokale import Statements (Punkt vor Dateiname),

```
from info_bme_reader import PickleReader, MnistReader
```

Testen Sie Ihre Abgabe über die Datei/das Script `assignment_1.py`.

2. Beschränkungen

- Fügen Sie keine nutzlosen Komponenten Ihrer Abgabe hinzu
- Sofern möglich, verwenden Sie eingebaute Funktionen
- Importieren Sie keine extra packages
- Alle Kommandozeilenparameter müssen optional sein.

3. Datei Header

All Ihre Quelldateien in Ihrer Abgabe müssen gleich zu Beginn einen Kommentar mit folgenden Informationen enthalten:

- Author: – Ihr Name
- MatNr: – Ihre Matrikelnummer
- Description: – Generelle Beschreibung der Datei

- Comments: – Kommentare, Erklärungen, usw

Bitte einfach den folgenden Code in Ihre Dateien kopieren und den Inhalt anpassen. Je nach PDF-Reader müssen Sie eventuell die Leerzeichen/Einrückungen per Hand anpassen.
Beispielheader:

```
##############################################################################
# Author:      Patrick Kasper
# MatNr:       0730294
# Description: The main file. Assignment only has 1 file...
# Comments:    This is the example comment. I just made it a bit
#              longer so it spans across multiple lines.
##############################################################################
```

4. Coding Standard

Für diese Lehrveranstaltung orientieren Sie sich am offiziellen PEP 8 Standard[8]. Dieser Beschreibt grundsätzliche Formalitäten im Bezug auf Ihren Code. Folgendes ist besonders zu Beachten:

Sprache. Code schreibt man in Englisch. Im internationalen Zeitalter ist es notwendig, dass auch jemand am anderen Ende der Welt verstehen kann, was Sie programmiert haben. Ihr gesamter Quellcode muss daher auf Englisch geschrieben sein. Dies betrifft sowohl die Kommentare also auch Variablennamen und Ähnliches.

Leerzeichen statt Tabulatoren. Python basiert auf Einrückungen, anstatt auf geschwungenen Klammern . Theoretisch gibt es die Möglichkeit, Leerzeichen (spaces) oder Tabulatoren (tabs) zu verwenden. PEP8 schreibt aber, 4 Leerzeichen als Einrückungen vor. Die meisten Python Programmierumgebungen werden automatisch 4 Leerzeichen einfügen, wenn Sie auf die Tabulator-Taste drücken.

Sprechende Namen. Verwenden Sie kurze, aber sprechende Namen für Ihre Variablen, Funktionen, (und Ähnliches). Es muss eindeutig aus dem Namen hervorgehen, was die Aufgabe des Elements ist. Für simple Iterationen kann ein einfaches i ausreichend sein, aber dies kann schnell zu Chaos führen. Sollten Sie Variablen haben, die keine

[8] `https://www.python.org/dev/peps/pep-0008/`

Aufgabe haben und nicht verwendet werden, schreibt PEP 8 vor, einen einfachen Unterstrich (_) zu verwenden. Sollten Sie sich unsicher sein, dann beschreiben Sie ihre Variablen (und Namen) in einem Codekommentar.

120 **Zeichen Zeilenlänge.** PEP8 sieht Zeilenlängen von maximal 79 Zeichen vor. Ein anderes, etwas großzügigeres Limit, welches sich in der Szene etabliert hat, sieht eine Länge von 120 Zeichen vor. In dieser LV verwenden wir daher das erweiterte Limit. Bitte achten Sie darauf, dass keine Zeile in Ihrem Code diese Länge von 120 Zeichen überschreitet (gilt auch für Kommentare). *Hinweis:* Zeilenlänge inkludiert die Einrückungen mittels Leerzeichen!

5. Automatisierte Tests

Ihr Programm wird mit dem folgenden Befehl ausgeführt:

```
>python assignment_1.py
```

Vergewissern Sie sich, dass Ihre Abgabe allen Beschränkungen, die in dieser Angabe erwähnt sind, konform ist.

6. Abgabe

6.1. Deadline

13. April 2018 um 23:59:59.

Eine Spätabgabe ist nicht vorgesehen. Ausnahme bilden hier Notfälle. Sollte das Abgabesystem nicht online sein, verlängert sich die Deadline automatisch um 24 Stunden. Sollten Sie, aus diversen Gründen, nicht in der Lage sein, Ihre Abgabe hochzuladen, kontaktieren Sie Ihren Tutor *VOR* der Deadline. (*Hinweis:* Urlaub oder Ähnliches wird nicht als Grund akzeptiert!)

6.2. Hochladen der Abgaben

Assignments werden stets als Archive abgegeben. Erlaubt sind hier die Formate *.zip*, und *.tar.gz*.
Zusätzlich zu ihren Quelldateien, soll Ihre Abgabe auf eine Datei namens *readme.txt* beinhalten. Das Vorhandensein der Datei ist Pflicht, ihr Inhalt aber optional. Sie soll folgenden Inhalt haben: (i) Die Zeit, die Sie benötigt haben, um die Aufgabenstellung zu absolvieren. (ii) Feedback, wo Sie Probleme hatten.
Abgaben erfolgen auf der Palme Website. Bitte prüfen Sie vor der Abgabe diese Kriterien:

- Datei- und Ordnerstruktue (siehe unten)
- Kommentarheader in jeder Quelldatei
- Coding Standard

6.3. Struktur der Abgabe

```
assignment_1.zip (or assignment_1.tar.gz)
    readme.txt
    assignment_1.py
    info_bme_reader
        __init__.py
        reader.py
        mnist_reader.py
        pickle_reader.py
```

Preliminaries

Appendix .1. Endianness

Die Endianness beschreibt, in welcher Reihenfolge die Bytes (nicht Bits) gespeichert sind. Simpel gesagt, ob man von Links nach Rechts oder von Rechts nach Links lesen muss. Im ersten Fall (little-endian, von links nach rechts) entspricht `00 FF` der Zahl 255. Liest man dies nun von der anderen Seite(big endian), ist es die Zahl 65280 (jeweils unsigned). Die Endianness ist in der Regel auf Hardware-Ebene definiert. Die meisten modernen Architekturen verwenden little-endian. Python übernimmt für Sie normalerweise jegliche notwendige Konvertierung. Wenn Sie aber mit Binärdateien arbeiten, kann es nun vorkommen, dass die Endianness für Sie eine Rolle spielt. Packages, die für derartige arbeiten ausgelegt sind (z.b.: struct) bieten daher fast immer eine Möglichkeit zur manuellen Kontrolle an. Wenn `struct.unpack()` zum Beispiel 4 gelesene Bytes zu einem Integer kombiniert, können Sie mit `'<','>'` diese Byte Order spezifizieren. MNIST Daten sind zum Beispiel in Big-Endian encodiert. Sie müssen daher `'>i'` verwenden, um den korrekten Integer zu erhalten.

Appendix .2. Samples, Features, und Labels

Im Bereich des Machine Learning wird sehr oft von Samples und Features gesprochen. Als Beispiel verwenden wir die Wettervorhersage (Regen am Abend, ja oder nein). Wir haben hier Sensoren, die die Temperatur, Luftfeuchtigkeit und die Dichte der Wolkendecke messen. Wir messen diese Werte jeden Tag um 12:00 Mittag. Jeder Datenwert (Luftfeuchtigkeit, Temperatur,...) gilt als Feature. Bei einem Sample handelt es sich um eine zusammengehörende Beobachtung bestehend aus deine Gruppe von Features. Ein Sample Besteht im Beispiel also aus einem Wert für Luftfeuchtigkeit, einem Wert für Temperatur und einem Wert für die Wolkendecke. Das Label schlussendlich beschreibt, welchen die wahre Klasse. Im Beispiel, ob es am Abend Regnet oder nicht. Eine der Hauptaufgaben von Machine learning ist es nun, die Kombination von Features und Labels zu lernen um dann für noch nie gesehene Samples eine Labelvorhersage treffen zu können. In der Community sind die Namen `X` (großes X) für die Features und `y` (kleines Y) für die Labels üblich. Daher werden Sie auch so in den Assignments verwendet, obwohl diese sonst auch dem Codingstandard widersprechen würden.