

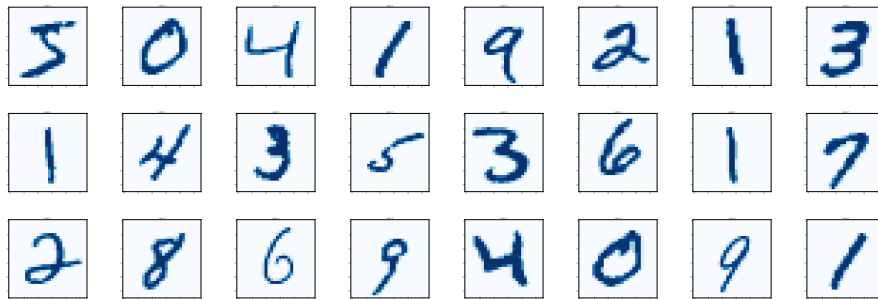
“Object Oriented Programming”

Assignment 1

Informatics 2 for Biomedical Engineering
Graz University of Technology

Abstract





The MNIST Database of handwritten digits¹ is a widely used dataset für klassifikation tasks. It consists of 65.000 samples of handwritten digits. In this first assignment, you will be tasked to implement a package that reads the dataset and prepares it for the future steps.



¹<http://yann.lecun.com/exdb/mnist/>

1. Dataset

Download the data files from the following website: <http://yann.lecun.com/exdb/mnist/>
In particular, you will need the following files:

- [train-images-idx3-ubyte.gz](#) 
- [train-labels-idx1-ubyte.gz](#) 
- [t10k-images-idx3-ubyte.gz](#) 
- [t10k-labels-idx1-ubyte.gz](#) 

2. Tasks

In this assignment you will write a python package consisting of 3 classes. One abstract base class and two derivatives. The file `assignment_1.py` is to use for testing of your own assignment. Use it to make sure your code works. Note that in all automated tests this file will be replaced.

2.1. Abstract Reader Class (4pt)

Import the package for abstract base classes. You can use the following code snippet:

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



Create a class called `Reader`. It should be a child of the `ABC` class.

2.1.1. Functions

`__init__(self, data_path)`: The constructor of the class should require one parameter defining the directory where the data files are being stored. Test here if the directory exists (and is in fact a directory). Else raise a `FileNotFoundError`.

`(abstract)read_data(self)`: The abstract method `read_data` is implemented by the concrete classes. Thus, in the abstract class the following function body is sufficient:

```
raise NotImplementedError
```



Use the decorator `@abstractmethod` to explicitly declare this as an abstract method. (Not that you need to import the `abstractmethod` from the `abc` package!)

`dump(self, dump_path)` The dump function should be able to store the data as a pickle. The parameter is the complete path (directory + file name). Make sure the directory exists and then create an object with the following structure and save it as pickle:

```
dump_data = {
    "X": None, # The features of the samples
    "y": None, # The labels of the samples
    "num_rows": None, # Number of rows per image
    "num_cols": None, # Number of columns per image
} # Replace the None with the actual values!
```

`plot_sample(self, index, store_path)` : This function is used to visualize a sample. Use the matplotlib `textttimshow` function². It require a list of lists (actually a numpy matrix). Use can create them from the features list with `np.split`³ or `np.reshape`⁴. Which sample you should visualise is defined via the first (index) parameter. Store the resulting image at the path defined by the second parameter.

Hint: use this function to test if your reader works correctly.

2.1.2. Properties

X: This property contains all datapoints (features) for all samples. It is a list of lists (again, you can use a numpy matrices here). Here is an example for the structure with the samples `s01` and `s02` each consisting of 4 features.

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

External writing of this variable should be blocked via the setter.

y: This variable contains the lables for all samples. As each sample only has one distinct label (the number represented on the image) the property `y` is a simple list.

num_rows: The number of data-rows per sample. As the sample is an image this is representative of the vertical resolution.

num_cols: The number of data-columns per sample. As the sample is an image this is representative of the horizontal resolution.

²https://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>

2.2. MNIST Reader (5pt)

The `MnistReader` class has the task to read, parse, and verify the original MNIST data. Override the `read_data()` method of the parent class. The parameter are two tuples. One for the features and one for the labels respectively. Each tuple consists of the file name and the associated *Magic Number*. Here is an example call of the function:

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

As a first step, verify if both files exist. To open the files (archives) you can use the `gzip` package. When you open them with `gzip.open()` the files will handle like any other file.

2.2.1. Reading the Data

Second, you will need to verify the magic number. To do so, read the first 4 byte of the file and compare their integer value with what has been passed in the parameter tuple. Should the numbers not match, raise an appropriate exception. To read the data you should use the `struct` package and the `struct.unpack`⁶ function. For integers use the Flag `>i`. (The `>` defines the endianness, the order in the which the bytes are stored.) The final check is the comparison of the number of sample in the samples and labels files. Here, open both files and jump to position 4 (you can use `read(4)`, or `seek(4)`). Read the next 4 byte (again, use the `>i` Flag) and compare the values.

Now load the labels. Open the file and jump to position 8 (to skip the magic number and the number of samples). Read the remainder of the file using the flag `'B'` (for unsigned char). The function `struct.iter_unpack()`⁷ could be very helpful here. Store the read labels in the associated property `y`.

Finally, load the features. Open the file and jump to position 8. The next two integers defined the number of rows followed by the number of columns. Use the same method you use before to extract the values and store them into the properties. Read the remainder of the file. For each

⁵<https://docs.python.org/3/library/gzip.html>

⁶<https://docs.python.org/3.5/library/struct.html#struct.unpack>

⁷https://docs.python.org/3.5/library/struct.html#struct.iter_unpack

sample read `num_rows * num_cols` values ('B' flag). Store this list of lists (one sublist for each sample. Or use a numpy matrix) in the property `X`.

Hint: Use private functions (`_` - Prefix). Examples: Verification of the magic number and the extraction of the labels and features.

2.3. Pickle Reader (3pt)

Create a second class derived from the `Reader` base class. The task of this class called `PickleReader` is to load and restore the data that were stored in pickle files via the `dump` function. To do this you need you override the `read_data` function of the parent class. The parameter is the path (the name) to the pickle file. (Remember that you already defined the directory in the constructor).

The data and properties `X`, `y`, `num_rows`, `num_cols` should be read and restored correctly. Do not forget to verify that the requested file actually exists!

Hint: When you created the concrete classes (`MnistReader` and `PickleReader`) that their constructor must call the constructor of the parent class (via `super().__init__()`). Else you will not be able to use the functionality contained in the parent class (for plotting and saving).

2.4. Package (3pt)

Implement your submission as python package with the name `info_bme_reader`. Make sure we can import your package with the following code. Create the `__init__.py` File and import these publicly available classes there.

Hint: Use relative import statement (dot `'.'` before the file name),

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



Use your script in `assignment_1.py` to test this.

3. Restrictions

- Do not add any bloat to your submission
- If possible, use built-in functions
- Do not import any packages that you do not
- Any command line parameters in your program must be optional

4. File Headers

All your source files in your submission must contain the following header consisting of a comment block with identifiable information.

- Author: – Your name
- MatNr: – Your matriculate number
- Description: – General description of the file
- Comments: – Comments, explanations and so on

Please just copy the following code directly into your files and change the contents. Note that, depending on your PDF-Reader you may need to manually fix the spaces/indentations.

Example Header:

```
#####  
# 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.  
#####
```



5. Coding Standard

This lecture follows the official PEP 8 Standard⁸. It defines basic formalities as to how your code should look like. In particular, please look at the following aspects:

Sprache. Programming is done in English. This is to ensure someone else at the other end of the world can read your code. Please make sure all sources you submit are thus written in English. This covers both variable names and comments!

Spaces, not tabs. Indent your files with 4 spaces instead of tabs. PEP 8 forces this in Python 3 (whilst allowing more freedom in Python 2). Most editors have an option to indent with 4 spaces when you press the tab button!

⁸<https://www.python.org/dev/peps/pep-0008/>

Descriptive names. Use descriptive names for variables where possible!

Whilst a simple *i* can be enough for a simple single loop code can become very messy really fast. If a variable has no purpose at all, use a single underscore (`_`) for its name. Should you be uncertain if a name is descriptive enough, simply as a comment describing the variable.

120 characters line-limit PEP 8 suggests a maximum line length of 79 characters. A different established limit proposes 120 characters. In this lecture, we thus use the wider limit to allow for more freedom. The maximum number of characters is 120 including indentations! Note that this is also applies to comments!

6. Automated Tests

Your program will be tested with the following call:

```
>python assignment_1.py
```

Remember that that this `assignment_1.py` file will be supplied by us. Vergewissern Sie sich, dass Ihre Abgabe allen Beschränkungen, die in dieser Angabe erwähnt sind, konform ist.

7. Submission

7.1. Deadline

13th of April 2018 um 23:59:59.

Any submission handed in too late will be ignored with the obvious exception of emergencies. In case the submission system is globally unreachable at the deadline it will be pushed back for 24 hours. If for whatever reason you are unable to submit your work, contact your tutor *PRIOR* to the deadline.

7.2. Uploading your submission

Assignment submissions in this course will always be archives. You are allowed to use .zip or .tar.gz formats.

In addition to your Python source files, please also pack a *readme.txt*. The file is mandatory but its contents are optional. In this file please write down how much time you spent on the assignment and where you ran into issues. Your feedback allows for immediate adjustments to the lecture and tutor sessions.

Upload your work to the Palme website. Before you do please double-check the following aspects:

- File and folder structure (see below)
- Comment header in every source file
- Coding Standard

7.3. Your Submission File

```
├─ 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
```