“Data Analysis and Preprocessing”

Assignment 2

Informatik 2 für Biomedical Engineering

Graz University of Technology

Abstract

Before we can actually start working with any kind of data, we have to investigate how it is constructed. This way we can make some predictions how well various algorithms will function and which preprocessing steps we have to take. In this assignment we will build upon the Reader package from Assignment 1 and create a class to calculate and visualise a range of characteristics.

---

1. Tasks

Write the class `InfoBme` (in the file `info_bme.py`). Mind the following specifications:

1.1. Reader Package

The Reader package from Assignment 1 should be available in the root directory of your python code. Do not submit the Reader when you hand in your assignment (it will be automatically included).

1.2. Class InfoBme(4pt)

1.2.1. Constructor

The constructor expects a correct instance of an `info_bme_reader` as param. This can either be the MnistReader or a PickleReader from the last assignment. Check if the passed object is indeed one of the valid classes. (**Hint:** You can use `isinstance() on the mother class.` Save the reader in a variable. Further, create a variable called `self.read_data` and assign the `read_data` function of the reader to it. This way you expose the function of the reader to anyone who uses your new InfoBme class.

1.2.2. Properties

`X:` Points to the property `X` of the reader.

`y:` Points to the property `y` of the reader.

1.3. (public) Functions

1.3.1. get_ class(label) - (1pt)

Returns the indices of all samples with the wanted label.

Param `label`: The label of the requested class.

Return (numpy array): The indices of the samples belonging to the requested class.

1.3.2. calc_ balance(indices) - (1pt)

Calculates the numbers of occurrence of each class for the requested indices.

Param `indices` (list, numpy array, str): The indices for the samples. If the string `'all'` is passed, calculate this across all samples.

Return (dict): Dict: key = Label/Class, value = Frequency(# Occurrences)

1.3.3. calc_ means(indices, plot=False, plot_ name=None) - (1pt)

Calculates the average feature-values of the samples (defined by the requested indices). Further it should be possible to visualize these mean values over all requested samples. In short, you create the mean image. The first pixel of the image is the mean of the first pixel for all samples. Use the mean and not a different averaging method (such as the median). To plot use `imshow()`[1] similar to the way you plotted in the readers). **Hint:** Get all samples of a specific class (via `get_class()` and calculate their means.

Parameter `indices` (list, numpy array): The indices from which you should calculate the means.

(KW) Param `plot` (bool): Defines if the function should visualize the data.

(KW) Param `plot_name` (str): If `'plot is True'` then this parameter defines the filename of the resulting plot. **Hint:** Raise an exception when the user requests a plot but fails to provide a name for the file.

Return (numpy array): The list of average values.

[1] https://matplotlib.org/api/_as_gen/matplotlib.pyplot.imshow.html

1.3.4. calc_hist(indices, plot=False, plot_name=None) - (1pt)

Calculate the histogram of a requested set of samples. Count how often each value occurs in the samples. If requested, visualize the histogram. Since we have concrete values it the function `fill_between()`[2] with the keyword parameter `step=True` can be very helpful.

Param `indices` **(list,numpy array):** The indices of the samples.

(KW) Param `plot` **(bool):** Defines if the function should visualize the data.

(KW) Param `plot_name` **(str):** If `'plot is True'` then this parameter defines the filename of the resulting plot. **Hint:** Raise an exception when the user requests a plot but fails to provide a name for the file.

Return (dict): Dict: key = value, value = Frequency (# Occurrences)

1.4. calc_bbox(indices) - (2pt)

Calculate the bounding box of the samples/images. It is defined by two values (row and column indices). P_0, where above and to the left all values are 0 and P_0, where all values below and to the right are 0. Remember that you will have to use the `num_rows` and num_cols Properties we have stored in the reader.

Param `indices` **(list, numpy array, str):** The indices of the samples for which we want to calculate the bounding box. If the string `'all'` is passed, calculate this across all samples.

Return (tuple): `((Row` P_0 `, Col` P_0`), (Row` P_1`, Col` P_1`))`

[2]https://matplotlib.org/api/_as_gen/matplotlib.axes.Axes.fill_between.html

1.4.1. calc_cosine_similarity(a, b, plot=False, plot_name=None) - (2pt)

Calculate the cosine similarity of two samples (a,b). It is defined as follows:

$$similarity = \frac{\sum_{i=1}^{n} a_i b_i}{\sqrt{\sum_{i=1}^{n} a_i^2}\sqrt{\sum_{i=1}^{n} b_i^2}} \tag{1}$$

a_i is the value of index i from the list of values of sample a.

When the user wants a comparison visualised, create a simple scatterplot[3]. (Use the values of `a` as param `x` and the values of `b` as param `y`).

Param `a` (numpy array) Sample a

Param `b` (numpy array) Sample b

(KW) Param `plot` (bool): Defines if the function should visualize the data.

(KW) Param `plot_name` (str): If `'plot is True'` then this parameter defines the filename of the resulting plot. **Hint:** Raise an exception when the user requests a plot but fails to provide a name for the file.

Return (float): The cosine similarity of the two samples..

[3]https://matplotlib.org/api/_as_gen/matplotlib.pyplot.scatter.html

1.4.2. generate_strat_splits(size) - (3pt)

To be able to classify in the future, we have to split the samples in sets. Namely in `training` and `verification` sets. It is important that each set contains a balanced set of samples and no sample is in more than one split. Write a generator that yields the indices of a training set per run.

Example: `size`=0.25
Data (labels): $[4, 2, 1, 3, 1, 4, 1, 2, 3, 4, 3, 4, 2, 1, 2, 3]$

- Indices split 1: $[0, 1, 2, 3]$
- Indices split 2: $[4, 5, 7, 8]$
- Indices split 3: $[6, 9, 10, 12]$
- Indices split 4: $[11, 13, 14, 15]$

Param `size` (float) *size∗num_samples* Fraction of samples per split (round up). **Hint:** The last set can/will be smaller. This is okay!

Return (numpy array): Indices of the current split/set. **Hint:** Python generators return their values with the command `yield` instead of `return`.

2. 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

3. 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.
##############################################################################
```

4. Coding Standard

This lecture follows the official PEP 8 Standard[4]. 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!

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

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!

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!

5. Automated Tests

Your program will be tested with the following call:

```
>python assignment_2.py
```

Remember that that this `assignment_2.py` file will be supplied by us. Use your version to test your code. Furthermore, ensure, all files you submit follow the rules and restrictions mentioned in this assignment.

6. Submission

6.1. Deadline

4^{th} of May 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.

6.2. 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

6.3. Your Submission File

```
assignment_2.zip (or assignment_2.tar.gz)
    readme.txt
    info_bme.py
    assignment_2.py
```

Important: Do ***NOT*** resubmit the reader package you wrote in assignment 1!