“Digit Classification with 1NN”

Assignment 3

Informatics 2 for Biomedical Engineering

Graz University of Technology

Abstract

Now that we have completed all preparations, this assignment tackles the task of classifying the handwritten digits. To evaluate the labelling we will utilise established metrics. Lastly, this assignment also confronts us with aspects relevant in team-oriented software development.

1. Tasks

Build ontop of your previous submissions. The package from Assignment 1 and the class from Assignment 2 should be available in the root directory. Here is the structure of your project:

```
│
■─ info_bme_classifier.py
■─ info_bme.py
■─ info_bme_reader
    │
    ■─ __init__.py
    ■─ reader.py
    ■─ mnist_reader.py
    ■─ pickle_reader.py
```

Important: Do when you hand in your work, do not re-submit the components from the previous assignments.

1.1. Classe InfoBmeClassifier

Create the file `info_bme_classifier.py` and implement the following class called `InfoBmeClassifier`.

Wichtig: Python packages such as SciPy or sklearn are not allowed in this task. You can use them to verify your results, but do not use them anywhere in this class. Numpy is allowed!

1.1.1. Constructor

Signature `__init__(self, info_bme)`

Parameter `info_bme`**:** An instance of the InfoBme classe from Assignment 2. Use it whenever you need to access data or implemented functionality.

1.1.2. Method classify (3pt)

This method does the classification. Create sets of data (utilising the `generate_strat_splits` form Assignment 2) and call the texttttnearest_neighbor function with every set. Finally re-combine the returned labels.

The return value of this function is a Tuple consisting of the merged Labels (called y_pred) and the actual ground truth (`y_true`). `y_true` is the data stored in `info_bme.y`. **Important:** Make sure that you do not jumble up the ordering of `y_pred` and `y_true`.

The parameters `k` and `sample_length` can be ignored for now. Simply pass the defaults to the classification algorithm.

If the user requests a plot (defined via the parameter), create a heatmap, where each field corresponds to the number of times a true label (y_axis) was classified as the predicted label (x_axis). **Hint:** When you do not include precise counts in your plot, you will have to either log-scale the values or eliminated all correctly matched samples (where `y_pred` equals `y_true`). Else your main axis will be so dominant, you wont see any misclassified samples.

An example for such a heatmap (also knows as *Confusion Matrix*):

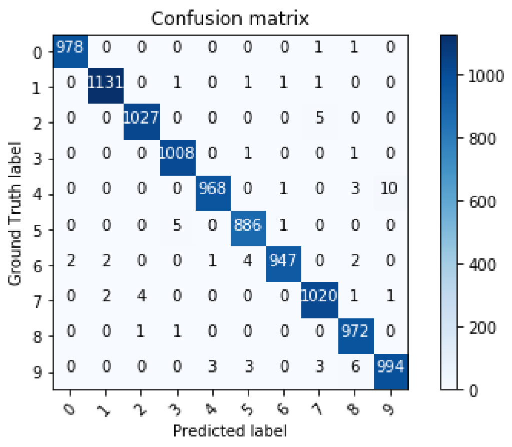

Image taken from Quora: https://www.quora.com/Classification-machine-learning-What-is-confusion-matrix

Signature `classify(self, split_size=0.2, k=1, sample_length=1.0, plot_name=None)`

Parameter `split_size` **(float):** Size of the splits for your k-folding

Parameter `k` **(int):** Param for the call of nearest_neighbor.

Parameter `sample_length` **(float):** Param for the call of nearest_neighbor.

(KW) Parameter `plot_name` **(str):** If used, then this parameter defines the path and name of the plot. If it is not passed (or is `None`), do not create the heatmap.

Return (tuple): y_pred (Labels of your classification), y_true (ground truth labels) (y_pred and y_true are numpy-ndarrays)

1.1.3. Method nearest_ neighbor (3pt)

This method implements a *k Nearest Neighbor* algorithm. The parameters are the indices of the samples to classify. For each sample you search for the most similar sample in the remaining data (any sample whose index is not in the set of passed indices) and store the label of this most similar sample. Use the cosine similarity from Assignment 2 to calculate the similarity of two samples. The parameters `k` and `sample_length` are only used for the bonus tasks.

The return value of this method is a numpy.ndarray consisting of the labels for your prediction.

Signature `nearest_neighbor(self, test_indices, k, sample_length)`

Parameter `test_indices` **(list/np.ndarray):** The indices of the current set.

Parameter `k` **(int):** See Bonus Task 2.1.

Parameter `sample_length` **(float):** Se Bonus Task 2.2.

Return (np.ndarray): y_pred

1.1.4. Method precision (1pt)

This method calculates the precision score of a classification. It is defined as follows:

$$P = \frac{T_p}{T_p + F_p} \tag{1}$$

For each class you calculate the following:

- T_p: True Positives, the number of correctly classified samples.
- F_p: False Positives, the number of samples you mistakenly classified as the current class.

To calculate the global precision score, you need to calculate the weighted average from the classes. The weights are the numbers of samples per class (based on the ground truth).

Signature `precision(self, y_true, y_pred)`

Parameter `y_true` (list/np.ndarray): The true labels.

Parameter `y_pred` (list/np.ndarray): Your predicted labels.

Return (tuple): global_precision_score, dict of *class: precision score for the class*

1.1.5. Method recall (1pt)

This method calculates the recall score of a classification. It is defined as follows:

$$R = \frac{T_p}{T_p + F_n} \tag{2}$$

For each class you calculate the following:

- T_p: True Positives, the number of correctly classified samples.
- F_n: False Negatives, Number of samples belonging to this class that you classified as something else.

To calculate the global recall score, you need to calculate the weighted average from the classes. The weights are the numbers of samples per class (based on the ground truth).

Signature `recall(self, y_true, y_pred)`

Parameter `y_true` (list/np.ndarray): The true labels.

Parameter `y_pred` (list/np.ndarray): Your predicted labels.

Return (tuple): global_recall_score, dict of *class: recall score for the class*

1.1.6. Method f1_score (1pt)

This method calculates the f1 score of a classification. It is defined as follows:

$$F1 = 2 \times \frac{precision \times recall}{precision + recall} \tag{3}$$

Hint: Use your functions for precision and recall. Then you do not need to worry about the weighted averages for the global score.

Signature `f1_score(self, y_true, y_pred)`

Parameter `y_true` **(list/np.ndarray):** The true labels.

Parameter `y_pred` **(list/np.ndarray):** Your predicted labels.

Return (tuple): global_f1_score, dict of *class: f1 score for the class*

1.2. Documentation (4pt)

Carefully document your code. To do this, follow the PEP 257 standard[1]. Each function and method has to have a docstring explaining the function, its parameters and its return values.

1.3. SVN (3pt)

To collaborate with your teammembers, use a subversion (SVN) project in the TUGonline. Add your tutor as reader to the project. A short introduction, how to create these projects and maintain them, can be found on the TU Graz infopage[2].

Important: Make sure you use clean and helpful commit messages Commit Messages.

1.4. Report (3pt)

Create a short report describing your results. What worked? Which samples were not correctly classified?. Try to reason why this happened. Use the plot functionalities that you have implemented throughout the assignments.

[1] https://www.python.org/dev/peps/pep-0257/
[2] https://svn.tugraz.at/

2. Bonustasks

2.1. KNN (4pt)

Extend the method `nearest_neighbor`. Previously you considered the single most similar sample. Now look at the k most similar ones (so if $k = 1$ you have no change). Use the already implemented parameter k to decide how many samples to consider. There are multiple methods on how to do the final label selection. For example, you can perform a simple majority vote based on the k most similar samples. A more sophisticated method is to weigh each of the k samples based on their ranking.

Experiment with different values for k and add your findings to the report.

2.2. Reduced Features (6pt)

Previously you used all features (all pixels) to calculate similarity and classify the samples. Now, simply reduce the number of samples (cut off the end) based on the fraction passed via the parameter `sample_length`). For example: Normally, all samples have 784 features (pixels). With $sample\_length = 0.5$ you would only use the first 392 ones.

Reduce the number of features step by step and observe the impact on the classification perforamce (precision/recall/f1). Add any findings to your report.

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 Standard[3]. 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!

[3]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!

6. Automated Tests

Your program will be tested with the following call:

```
>python assignment_3.py
```

Remember that that this `assignment_3.py` file will be supplied by us. Use your version to test your code or create visualisations for your report. Furthermore, ensure, all files you submit follow the rules and restrictions mentioned in this assignment.

7. Submission

7.1. Deadline

8^{th} of June 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_3.zip (or assignment_3.tar.gz)
    readme.txt
    report.pdf
    assignment_3.py
    info_bme_classifier.py
```

Important: Do ***NOT*** resubmit any component of previous assignments!