CS-233: Introduction to Machine Learning

Milestone 2

Welcome to the project-MS2 for the course Introduction to Machine Learning. Our goal will be to implement a few of the methods seen in the course and to test them on a dataset. We will be evaluating them on evaluation metrics specific for the tasks and present our results with a project report at the end of each milestone.

As deliverable, you are expected to fill a code framework: python files `.py` that implement these specific methods, and then a main python script `main.py` to call them on the data. We will use a subset of the DermaMNIST dataset, which is a small dataset of dermoscopic images of skin lesions across seven diagnostic categories: the goal is to output the correct diagnostic for each image.

You can find more information in the respective chapters below.

Important Information:

- This will be a group project and must be done in **groups of 3.** Please inform us if you are not 3.
- You may post your questions about the project on the "Student Forum" on Moodle.
- *You must implement all the code by yourself*. You are not allowed to import and use external libraries (e.g., scikit-learn), unless it is *explicitly* said otherwise.
    - NumPy, PyTorch, and matplotlib are fine to use.
- **Deadline**:
    - Milestone 2: 01.06.2025 (23:59)

Grading

MS2 is also worth 10% of your final grade and will be graded as following: (**10pt**)

- Code (**3pt**): the framework code was filled as asked and works.
- Clarity and presentation (**3pt**): the report is clear, well presented and structured.
- Discussion (**4pt**): discussion of the work, what you did, what worked and didn't, how you made your choices, reporting of the results, comparison of methods, etc.

- *Creativity* (**1pt**): if you went beyond what is asked, e.g., by implementing something more, an interesting analysis or experiment, an especially well written code, etc.
    - Note: while you can earn up to 11pt, each milestone will be capped at 10pt.

Respect of the page limit: We will remove 1pt if the report does not respect the page limit.

The Framework

Structure

The framework is organized with the following folder structure: (you can download it from Moodle)

```
<sciper1>_<sciper2>_<sciper3>_project/
    main.py
    test_ms2.py
    report.pdf
    src/
        __init__.py
        data.py
        utils.py
        methods/
            __init__.py
            deep_network.py
            dummy_methods.py
```

- Please do not rename these files, nor move them around.
- Your report will also be a part of this folder structure (as `report.pdf`).
  Please include it in your framework when you are preparing your submissions. You will be submitting the zipped "project" folder.
- You can add your dataset inside the project folder (you will download it from moodle), *but make sure you remove it for the submission*, otherwise the file size will get very large!
- Make sure to take a look into `src/utils.py` for useful functions you could use.

Implementation

Some parts of the code are provided to help you get started. It will be your task to fill in the missing parts to get all the methods running on your data. You have to 1) fill the python files implementing the methods and 2) write the main script that runs these methods on the dataset.

- **Methods:** The methods are all implemented in the form of Python classes, under `src/methods/`. We have provided the implementation of a "dummy" classifier that returns random predictions as an example, in `src/methods/dummy_methods.py`. Study this class well, you can use it as templates to code your own methods. It contains some essential functions:
    a. `__init__(self, ...)`
        - This function is used to initialize an object of the class. You can give it arguments that will be saved in the object, such as the value of hyperparameters.

    b. `fit(self, training_data, training_labels)`
        - This is the training or fitting step of the model. Training data with corresponding labels are used to estimate the parameters of the model. This method should be called before the `predict` function.

    c. `predict(self, test_data)`
        - It generates predictions for the given data. It should be called after the `fit` function (you cannot predict without a trained model!).
- **Main script:** Contrary to the notebooks of the exercise sessions, we run python as a script in the terminal (see below for details on how to do it). You need to complete this script `main.py` to apply the methods to your data and evaluate them. Some pointers are given in the form of comments in the file. In brief, you are expected to follow these general steps:
    a. Loading the data
    b. Preparing / pre-processing the data
    c. Initialize the selected method
    d. Training and evaluating the selected method
    e. Reporting the results
- This script can be broken down in two main parts:
    a. The main function `def main(args):` which is where we write the code we run.
    b. And some code starting with `if __name__ == '__main__':`. This is some pythonic way of coding “run the following instructions only if this file is a

script". This is where we parse the arguments that were given in the command line and then call the `main()` function.

Running the Project

You will be running the Python scripts from your terminal. For example the main script can be launched from inside the project folder as:

```
python main.py --data <where you placed the data folder> --method dummy_classifier
```

You can specify the method and many other arguments from the terminal by writing them as `--arg value` in the command above. The arguments are defined at the bottom of the `main.py` script. We encourage you to check how they work, what their default values are, and you can even add your own!

Test Script

We also provide you with a test script `test_ms2.py`. This script is for you to verify your project folder and code structure, as well as testing your method on some *very easy* cases.

- These easy cases are by far not exhaustive and you should verify the correctness of your code by your own means as well!
- The script can be launched from inside the project folder as
    - `python test_ms2.py`
- By default, it hides any `print()` from your code. You can optionally pass it the argument `--no-hide` to re-enable the printing.

Milestone 2

For Milestone 2 (MS2), you have to implement:

- Deep Networks with PyTorch
    - MLP and CNN
- The main script that calls them.

And you will also need to write a **2-pages report** about what you did.

- **Deliverable:** You need to submit **your code** and a **2-pages report** (zipped together). The project report should be part of your folder structure (see above!)
    - You should name the project folder (replacing the `<sciper#>` by your own scipers) `<sciper1>_<sciper2>_<sciper3>_project`, and the zipped file should have the name `<sciper1>_<sciper2>_<sciper3>_project.zip`. Please make sure that when you unzip it, it extracts the folder `<sciper1>_<sciper2>_<sciper3>_project`.
- **Code:** You will need to complete the relevant parts of the following python files for this milestone. Please look inside the files for more information.
    - `main.py`
        - The main python script we use to run our methods. It works like the `main.py` script of MS1. You will need again to prepare the data, validation set, etc. then fill out the parts calling the deep network methods.
    - `src/methods/deep_network.py`
        - There is 3 class to fill using PyTorch, two models and a trainer:
            1. an `MLP` model, which should not use any convolutional layer and take vectors as input,
            2. a `CNN` model, which should use at least one convolutional layer and take images as input,
            3. and the `Trainer` class that is used to fit and predict with a deep network model.
        - For the `Trainer`, you will need to complete the functions `train_one_epoch()` and `predict_torch()`. We have already provided the functions `fit()` and `predict()` that serve as an interface between NumPy and PyTorch, so that we can call the trainer in the main script like the other methods from MS1.
- **Data:** We will be using the DermaMNIST dataset for MS2.
- **Report:** You need to write a concise *2-pages* report about your work. The project report should include your methodology and a brief summary of the results you have achieved with the methods.
    - We advise to follow a structure like “Introduction, Method, Experiment/Results, and Discussion/Conclusion”.

    - In Methods, you should explain your data preparation pipeline, what was your process to select hyper-parameters, describe your models and what you tried (layers, dimensions, …), and anything special you may have done.
    - For the results, how do the models perform with respect to hyper-parameters and architecture choices? You should also report the speed of the training/inference.
        - You can, and probably should, use graphs and/or tables.
    - Finally, discuss what you obtained, explain any surprising results you may have had, which method is best and why you think it is so.
        - Compare your final models (MLP, CNN) on the *test* data.
    - Stay concise and respect the page limit. You can use double column if needed.
- **Test:** Make sure that `test_ms2.py` runs without any problems! This ensures that your code compiles and your implementations can be imported.
    - And make sure that your project is running when you call `main.py` for each of the methods.The following commands should run without crashing:

Some additional points:

- Make sure that your method classes **do not modify the data** they are passed. If you are doing some form of data augmentation (such as adding a bias term, doing feature expansion etc.), then you must do so *outside* these classes, in your `main.py` script.
- Make sure your implementations of these methods are not dataset specific–they should work for arbitrary numbers of samples, dimensions, classes etc.
- Make sure that your project is running when you call `main.py` for each of the methods. In other words, the following commands should be running without crashing.

Sample commands to run:

CNN (with lr=0.00001 and max_iters=10 in that example)

```
python main.py --data <where you placed the data folder> --nn_type cnn --lr
1e-5 --max_iters 10
```

MLP (with lr=0.00001 and max_iters=100 in that example)

```
python main.py --data <where you placed the data folder> --nn_type mlp --lr
1e-5 --max_iters 100
```

There will be *no* competition this year.
You can add the `--test` argument to predict on the actual test set, otherwise your code should use a validation set!

Sample results

For the deep networks, we won't provide sample results; try to get the best performance you can with either model.

The DermaMNIST dataset (For MS2)

The DermaMNIST dataset contains dermoscopic images of 7 categories of skin lesions. Derived from the HAM10000 dataset, DermaMNIST serves as a standardized collection of dermatological images for machine learning applications in medical diagnosis. The dataset consists of 9,012 images divided into a training set and a test set, with each image resized to 28×28 pixels, with 3 color channels. Below is the number of images for each class.

Label	**Description**	**Count**
0	Actinic keratosis	294
1	Basal cell carcinoma	462
2	Benign keratosis	989
3	Dermatofibroma	103
4	Melanoma	1,002
5	Melanocytic nevus	6,034
6	Vascular lesion	128

The dataset is split into 7,007 train images and 2,005 test images. The data.py file contains a helper function load_data() to load the images and the corresponding labels (for classification) of the skin lesions. The labels have been cast into integers in {0,1,...,6}. The function returns as np.array:

- train_images of shape (7007, 28, 28, 3)
- test_images of shape (2005, 28, 28, 3)

- train_labels of shape (7007,)
- test_labels of shape (2005,)

The Tasks and Metrics

To measure the performance of our methods, we can use the following metrics.

Classification Metrics

Accuracy

We will report the average accuracy in percentage. It can be written as:

$$\mathrm{accuracy} = \frac{\text{number of correct predictions}}{\text{total number of predictions}} \times 100$$

Generally speaking, in case of imbalanced datasets (meaning that the dataset is not divided evenly among the different class labels), the accuracy metric can be too biased toward the largest classes.

Therefore, we will also report the F1-score, which may be more meaningful than the accuracy metric in these cases.

F1-Score

We will report the **macro F1-score**, which is an average of class-wise F1-scores.

The F1-score for a class *i* is defined as follow:

$$\mathrm{F1_i} = \frac{2 \times (\mathrm{P_i} \times \mathrm{R_i})}{\mathrm{P_i} + \mathrm{R_i}}$$

where $\mathrm{P_i}$ and $\mathrm{R_i}$ are the precision and recall values for the class *i*. Such precision and recall are computed as

$$\mathrm{P} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}$$

$$\mathrm{R} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}$$

- *True Positive*: true class is *i* and predicted class is *i*
- *True Negative*: true class is *not i* and predicted class is *not i*
- *False Positive*: true class is *not i* and predicted class is *i*

- *False Negative*: true class is *i* and predicted class is *not i*

Precision relates to how many of our predictions as class *i* are correct, while Recall relates to how many of the true samples of class *i* were predicted as such.

For K classes, we can compute a macro F1-score as:

$$\mathrm{macro\,F1} = \frac{1}{K}\sum_{i=1}^{K}\mathrm{F1_i}$$

Higher F1-scores correspond to a better model and macro F1-score reflects the model's average performance on all the classes.

Runtime Analysis

It is generally very useful to report how fast your algorithms can be trained and how fast they predict. You can easily measure the time passed in Python using the `time` module. Here is a quick example:

```
import time
s1 = time.time()
dummy_function()
s2 = time.time()
print("Dummy function takes", s2-s1, "seconds")
```

In case you get an error for import cv2

```
Install:
pip3 install opencv-python==4.6.0.66
```