Last edit: 18.03.2023

CS-233(b): Introduction to Machine Learning Course Project

Welcome to the project 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 HASYv2 dataset, which is a small dataset of hand drawn mathematical symbols: the goal is to recognize which symbol is drawn on the image.

You are encouraged to complete the project as you encounter the methods in the course. You will have two weeks before each milestone deadline where the exercise session will be dedicated to discussing the project with the assistants.

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.**
- You may post your questions about the project on the "Student Forum" on Moodle.
- The exercise sessions of weeks 6&8 and 13&14 can be used to make progress on your projects and ask your questions to the assistants.
- *You must implement all methods by yourself*. You are not allowed to import implementations of methods (e.g., from the scikit-learn library), unless it is *explicitly* said otherwise.
- **Important deadlines**:
    - Milestone 1: 21 April (23:59)
    - Milestone 2: 2 June (23:59)

Grading

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

- Code (**3pt**): the framework code was filled as asked.
- 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.

You can find more information about each Milestone in their respective chapters.

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_ms1.py
    report.pdf
    src/
        __init__.py
        data.py
        utils.py
        methods/
            __init__.py
            dummy_methods.py
            kmeans.py
            logistic_regression.py
            svm.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 Milestone 1 and Milestone 2 submissions. You will be submitting the zipped "project" folder.
- 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 follows 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_ms1.py` or `test_ms2.py`, depending on the milestone. 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_ms1.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 1

For Milestone 1 (MS1), you are expected to have running:

- K-Means,
- Logistic regression,
- SVM,
- The main script that calls them.

And you will 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 `<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. While the data is loaded for you, you should prepare it correctly (e.g., normalization, bias, validation set,...), and then call the methods to train and predict.
    - `src/methods/kmeans.py`
        - Implement K-Means, as seen in the exercises. You can play with the number of clusters K. While this is a clustering method, we can use it to make some predictions, similarly to what was done in the exercise!
        - We advise to also fill a function `k_means(self, data, max_iters)` that performs the K-Means algorithm, and to use it in the other functions.
    - `src/methods/logistic_regression.py`

        - Implement *multi-class* logistic regression. The training will use gradient descent to learn the weights. You can search for the best learning rate and/or the best number of max iterations.
        - Note: if you get `nan` when training, it is possible that your learning rate is too high and causes the weights and/or `np.exp()` to diverge.
    - `src/methods/svm.py`
        - **Exceptionally**, this one can be implemented using the scikit-learn implementation `sklearn.svm.SVC`. You should look into the different kernels and their parameters (at least *two* kernels amongst `linear`, `rbf`, and `poly`!).
- **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".
    - Points to consider when writing your report: your data preparation, the methods you tried, their performances, how you selected the hyperparameters, how do the *final* models compare on the test data, etc.
    - You should explain what hyperparameters you tried for these methods and what results you achieved. For instance, you can show simple visualizations of your validation performances with respect to the different hyperparameters.
    - Stay concise and respect the page limit. You can use double column if needed.
- **Test:** Make sure that `test_ms1.py` runs without any problems! This ensures that your code compiles and your implementations can be imported.

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:

K-Means (with K=3 in that example)

```
python main.py --data <where you placed the data folder> --method kmeans --K 3
```

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

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

SVM (with C=1, an RBF kernel, and gamma=0.01 in that example)

```
python main.py --data <where you placed the data folder> --method svm --svm_c
1. --svm_kernel rbf --svm_gamma 0.01
```

You can add the `--test` argument to predict on the actual test set, otherwise your code should use a validation set!

Sample results for each dataset:

Here are sample results that we obtained for these methods on the HASYv2 dataset.
Note that you should not necessarily get the exact same number, and that you can even get better ones for different hyperparameter values! You can use these results to sanity-check your implementations: if you have significantly worse results, then you might have a bug or bad hyperparameter values.

	K-Means	Logistic Regression	SVM
HASYv2	80% acc, 0.81 F1-score	89% acc, 0.89 F1-score	97% acc, 0.96 F1-score

Milestone 2

We will update this part of the document once we conclude Milestone 1 and Milestone 2 is ready. Similarly to MS1, you can expect to implement some methods, try them on a dataset, and write a report.

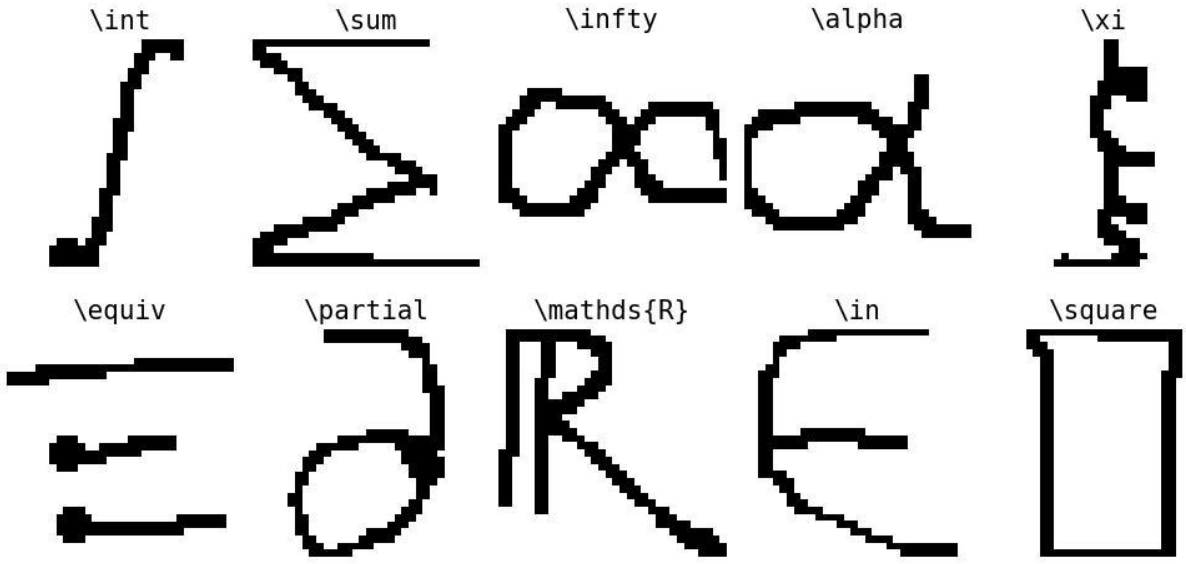
MS2 will be about dimensionality reduction (PCA) and deep learning.

The HASYv2 dataset

This dataset contains 168 233 images dispatched into 369 classes. The images are LaTeX symbols drawn by hand and the classes are their corresponding prompt to display them in a LaTeX format (ex. '\alpha' for the greek letter alpha). For the purpose of this project, only a sub-sample of the 10 most frequent classes are kept. They are:

- The integral operator `\int`
- The sum operator `\sum`
- The infinity symbol `\infty`
- The greek letter alpha `\alpha`
- The greek letter xi `\xi`
- The equivalence symbol `\equiv`
- The partial derivative symbol `\partial`
- The real number letter `\mathbb{R}`
- The set inclusion symbol `\in`
- The d'Alembertain operator (or "end of proof" symbol) `\square`

Below is an example of images for all the classes taken from the training set.



Label	\int	\sum	\infty	\alpha	\xi	\equiv	\partial	\mathbb{R}	\in	\square

Count	350	339	291	259	256	251	240	237	229	217

- Each image is 32x32 pixels
- The dataset his split into 2138 train images and 531 test images
- The `data.py` file contains a helper function `load_data()` to load the images and the corresponding labels that have been casted into integers in {0,1,...,9} following the order above. The function returns as `np.array`:
    - `train_images` of shape (2138, 32, 32)
    - `test_images` of shape (531, 32, 32)
    - `train_labels` of shape (2138,)
    - `test_labels` of shape (531,)

The Tasks and Metrics

For MS1, we will be working on classification. As discussed above, our subset of HASYv2 consists of small 32x32 images spread into 10 classes. 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:

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

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:

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

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")
```