# Assignment 4 - Classification benchmarks

**Peter Thramkrongart and Jakub Raszka**

## Github link

Link to the repository: <https://github.com/JakubR12/cds-visual.git>

Link to the asssignment folder: <https://github.com/JakubR12/cds-visual/tree/main/assignments/assignment-4>

## Contribution

Both Peter Thramkrongart and Jakub Raszka contributed equally to every stage of this project from initial conception and implementation, through the production of the final output and structuring of the repository. (50/50%)

## Description

In this assignment we created two command-line tools which can be used to perform a simple classification task on the MNIST data and print the output to the terminal. These scripts can then be used to provide easy-to-understand benchmark scores for evaluating these models.

We created two python scripts and an extra utility script. One takes the full MNIST data set, trains a Logistic Regression Classifier, and prints the evaluation metrics to the terminal. The other should take the full MNIST dataset, train a neural network classifier, and print the evaluation metrics to the terminal.

## Methods

For modeling, we used the LogisticRegression() classifier from sci-kit learn, and the NeuralNetwork class that Ross had build from scratch using only Numpy. They both use a custom function from the utility script to preprocess the data. It fetches the data, rescales it, and train/test splits it.

## Results

Our initial Logistic Regression baseline had a weighted average accuracy of 92%. By comparison, using the neural network model resulted in an accuracy of 96% after 15 epochs. The NeuralNetwork() class was built in pure numpy, and lacked features for finding the optimal architecture and stopping point for training. We suspect that an even higher accuracy can be reached with this model, if you are lucky with the architecture and parameters.

Code wise, we could also improve our scripts so that no time is wasted preprocessing data again and again.

Here are the metrics:

**Neural Network:**

precision recall f1-score support  
 0 0.95 0.98 0.97 3432  
 1 0.98 0.98 0.98 3947  
 2 0.95 0.96 0.95 3500  
 3 0.96 0.93 0.95 3615  
 4 0.96 0.95 0.95 3442  
 5 0.94 0.95 0.94 3156  
 6 0.97 0.97 0.97 3472  
 7 0.97 0.96 0.96 3639  
 8 0.94 0.95 0.94 3406  
 9 0.94 0.94 0.94 3391  
  
accuracy 0.96 35000  
macro avg 0.96 0.96 0.96 35000  
weighted avg 0.96 0.96 0.96 35000

**Logistic Regression:**

precision recall f1-score support  
 0 0.94 0.97 0.96 1361  
 1 0.95 0.97 0.96 1531  
 2 0.93 0.90 0.91 1477  
 3 0.91 0.90 0.90 1382  
 4 0.92 0.93 0.93 1371  
 5 0.89 0.88 0.88 1283  
 6 0.95 0.96 0.95 1396  
 7 0.94 0.93 0.94 1425  
 8 0.90 0.89 0.89 1373  
 9 0.90 0.90 0.90 1401  
  
accuracy 0.92 14000  
macro avg 0.92 0.92 0.92 14000  
weighted avg 0.92 0.92 0.92 14000

## Reproducibility

**Step 1: Clone repository**  
- open a linux terminal - Navigate the destination of the repository - run the following command

git clone https://github.com/JakubR12/cds-visual.git

**step 2: Run bash script:**  
- Navigate to the folder “assignment-4”.

cd assignments/assignment-4

* We have written a bash script *benchmark\_classifiers.sh* to set up a virtual environment, run the python scripts with default values, save the metrics, and kill the environment afterwards:

bash benchmark\_classifiers.sh

**Other options** You can also import the scripts submodule of the src main module in a jupyter notebook, or run the scripts from a the terminal. These ways you can make use of the parameter arguments to play with the model parameters. If you run the scripts directly from the terminal, you can use the create\_benchmark\_class.sh script to create the environment beforehand.

**Parameters for running lr\_mnist.py from the terminal**

There are 7 arguments which can but do not have to be specified:

flags: -tr, --train\_size, default: 0.8: description: int or float, a proportion of the data the model to be trained on  
  
flags: -ts, --test\_size: default: 0.2, description: int or float, a proportion of the data the model to be trained on  
  
flags: -r, --random\_state: default: None, description: int, a random seed  
  
flags: -sm, --save\_metrics: default: None, description: bool, whether to save the metrics  
  
flags: -mf, --metrics\_safe, default: logistic\_regression\_metrics.txt, description: str, the filename of the metrics with the .txt ending  
  
flags: -t, --tolerance, default: 0.1, description: float, Tolerance for stopping criteria  
  
flags: -p, --penalty, default: None, description: "none" "l1", ‘l2’, "elasticnet

example:

python lr-mnist.py -tr 0.7 -ts 0.3 -r 2 -sm -mf log\_reg\_filename.txt -t 0.001 -p l2

When using boolean flags (-sm), just leave it empty.

**Parameters for running nn\_mnist.py from the terminal**

There are 7 arguments which can but do not have to be specified:

flags: -tr, --train\_size, default: 0.8: description: int or float, a proportion of the data the model to be trained on  
  
flags: -ts, --test\_size: default: 0.2, description: int or float, a proportion of the data the model to be trained on  
  
flags: -r, --random\_state: default: None, description: int, a random seed  
  
flags: -sm, --save\_metrics: default: None, description: bool, whether to save the metrics  
  
flags: -mf, --metrics\_safe, default: neural\_network\_metrics.txt, description: str, the filename of the metrics with the .txt ending  
  
flags: -e, --epochs, default: 15, description: int, a number of epochs  
  
flags: -hl, --hidden\_layers, default: [32,16], description: list, a number of nodes in the two hidden layers, the third is set to 10 automatically

example:

python nn-mnist.py -tr 0.7 -ts 0.3 -r 2 -sm -mf neural\_network\_filename.txt -e 10 -hl 28, 14

When using boolean flags (-sm), just leave it empty.

## Running the project on something else than Linux

Our projects are mainly made for Linux/mac users. Our python scripts should run on any machine, though our bash scripts may not work. For this case, we recommend using the python distribution system from <https://www.anaconda.com/> to setup environments using our requirements.txt files.