Pattern recognition - Group tasks report

Students : Sergiy Goloviatinski, Ludovic Herbelin, Hina Khadija, Raphaël Margueron

Group organization

We mostly communicated in our group by Discord. For each task we separated it in sub-tasks and each group member decided on which sub-task he wanted to work.

Git was a great tool to share our progress on the tasks. Every time someone finished a specific part we push the work on the GitHub repository and everyone was able to see the code and work further on the task.

What worked, what did not work

Since the workload was really light for a group of 4 people we managed to complete every assignments.

When some people in the group had difficulties doing specific task we always found a way to overpass them by sending a message to our teammates.

Tasks

MNIST Dataset

We had to classify use the following classifier on the MNIST dataset:

- Support vector machine
- Multi-layers perceptron
- Convolutional neural network

For the convolutional neural network, we also had to test the classifier on a permutated version of the dataset.

Support vector machine (SVM)

The SVM classifier was implemented using the *scikit-learn* library.
We used the grid search algorithm (with **cross-validation**) in order to find the best hyperparameters.

However, since scikit-learn works only using CPU, we could not test many of the parameters, as it would take too much time to compute.

We have found the following hyperparameters to be the best for our data : `{'C': 1.0, 'degree': 2, 'gamma': 1.0, 'kernel': 'poly'}` . Using this, we achieved an accuracy of **98.06%** on the test set.

Multi-layers perceptron (MLP)

For this classifier, we used the *scikit-learn* library. Like for SVM, we opted to use a grid search approch to find the best hyperparameters.

The best hyper-parameter found were :
`{"activation": "logistic", "hidden_layer_sizes": 100, "learning_rate": constant", "learning_rate_init": 0.1, "max_iter": 550, "solver": "sgd" }`

Using these parameters we archived an accuracy of **97.16** on the test set.

Convolutional neural network (CNN)

We used the *pytorch* library to implement the CNN.
The NN architecture was quite simple:

- Conv layer: 28x28x3 -> 9x9x32
- leaky ReLU
- Conv layer: 9x9x32 -> 3x3x64
- leaky ReLU
- Conv layer: 3x3x64 -> 1x1x128
- leaky ReLU
- a flattering layer
- a 128 -> 10 linear layer for classification

Convolutional neural network (CNN) with Permutated dataset

For the permutated version of the MNIST classifier, we actually used the exact same CNN architecture as for the non-permutated dataset. We achieved **95.61%** of accuracy on the test set.

Keyword spotting

The image binarization part was done with Otsu's algorithm using the *openCV* python library. We also tested a binarization method where we had to manually specify the tresholds with *openCV* but the result was more deceiving. Another method that we tried was with the *skimage* library using Sauvola algorithm but the result was again worse than with Otsu's algorithm.

After the binarization part we had to extract features on the these images we used the following metrics :

- The number of transitions white -> black, black -> white
- The upper contour matrix
- The lower contour matrix
- The ratio of white pixel on the image

With these metrics we created a large vector for each words then, we computed the distance of every sample in the test set to the samples in the train set.

The distances were computed using the dynamic-time-warping algorithm (with the euclidian distance function). We used a python implementation called *fastdtw*.

After that we evaluated our system using the precision-recall curve our results were pretty bad with a maximal recall of **14%**

We think that this result is due to the fact that we didn't correctly preprocess the labels. So these strings : *"that's"*, *"thats"* were not equivalent in our case. We learned this later.

Signature verification

- input data is read-in by `parse.py`
- features are calculated using `features.py`
- features are normalized using `features.py` , where each feature is normalized only w.r.t. a single signature (see `features.Features#normalize_signature_features`)
- `dtw.py` imported from assignment 3 as-is, converting the two input series (s, t) to numpy arrays
- can represent a user by creating a new `user.User` object
  - takes a user id and an enrolment features dict as arguments. User id should be the first part of the corresponding set of signatures, e.g. user "007" represents signatures of the form "007-g-01.txt" or "007-12.txt".
  - enrolment features should already be normalized
  - during construction, new User object automatically collects its enrolment signatures from the enrolment features dict, according to its user id
  - User object can calculate dissimilarities of some target signature w.r.t. all its enrolment signatures (using `dtw.py`) (check `user.User#calculate_signature_dissimilarity`)
- `output.py` has a function to print the dissimilarity dictionary for a user and a target signature
- `runner.py` currently orchestrates
  - loading data
  - calculating features
  - normalizing features
  - creating a User object for each line in `SignatureVerification/users.txt`

    - demonstrating the calculation and output of a dissimilarity dict for a specific user and target signature
- `parse.py` has a method `get_ground_truth` to load a ground truth dictionary for the verification signatures. Not used yet.

Graph matching

We used the python library *networkx* to store and handle the graphs for this exercise, the main advantage was that this library provided us an implementation for the GED. After computing the distance matrix that represents the GED between each graph pair, we passed it to the *sklearn*'s KNN implementation.

The use of the *networkx* library greatly simplified our work.

General thoughts about the group exercise

We have found that since there is no grade for the exercises and they are only required to be able to take the exam, they were not really motivating.

Maybe they could contribute to the grade of the exam, even if the students need to achieve a note >= 4.0 for the exam it could help motivating us.

Another point was that the amount of work for the group tasks was small for a team of 4-5 people.

Also, because most students in our group took the Advanced Topics in Machine Learning course, we had already worked on more advanced deep learning methods, which contributed to the “small tasks“ problem. Though, this one is a bit inevitable since there might be students who do not.