Group Projects Report

Team 4.5 biologists

Team Organization

To organize our group we had brief discussions on site at the university after the lectures about the overall problem. Then we split the task into subtasks on the project manager on Github. During the work at home, we communicated with Slack or by video conference. This was very helpful to clarify appearing problems and for the further coordination of the task.

The group members with higher programming experience were helping and sharing their knowledge very well (e.g. exploring the use of Github) and everybody in the group was highly motivated to contribute something to the Exercises.

Sometimes it was difficult to keep track of the work of other people (much of code which was in some cases difficult to follow).

Projects Tasks

Please see the project README.md for more details.

Support Vector Machine

For the Support Vector Machine task, we choose the linear and the RBF kernel functions. Since the MNIST data can already be loaded in the binary format we did not need to spend any time in preprocessing or formatting. Since the MNIST data set contains 60'000 training samples and 10'000 test samples, we first used a small fraction of it to come up with working code. This approach is less time consuming when testing the code. The reduced data set was also used during parameter optimization (because grid-search is very slow).

For parameter optimization, we used the GridSearchCV function from sklearn.model_selection which gives a mean score (calculated via cross-validation) for every coordinate on a predefined grid. A reasonable range for the grid was chosen by simply trying different ranges and looking at the score of the cross-validation.
For the linear kernel, the parameter C needed to be optimized and for the RBF kernel, an additional parameter gamma was optimized via grid-search. The optimized parameters were

then used to build a model for the full training data set using the svm.SVC() function from sklearn.
The model was then tested with the full test data set.

Multi-Layer Perceptron

We used pytorch for all neural network implementations.

We built a 1 hidden layer MLP (nn.Linear) and using ReLU as an activation function. Since the input layer takes 28*28 data points (shape of the image), the width of the hidden layer is one of the parameters that needed to be optimized. Other parameters that needed to be optimized were the batch size (numbers of training examples in one iteration), the learning rate (which multiplies the gradients during backpropagation) and the number of epochs (one epoch is when all the image are passed once through the network, forward and backward).

Convolutional Neural Network

Convolutional Neural Networks surpass MLP in certain tasks, in particular when it comes to processing or linking spatial information. In addition, CNNs run faster than MLPs as they are generally “lighter” (less internal parameters).

We used a single convolutional layer, reducing 28x28 to 24 channels of 8x8, followed by a LeakyReLU. No maxpooling was performed as the output was directly flattened and then fed through a single fully connected layer for classification.

Both neural networks were tested on the MNIST and the permutated MNIST datasets.

Overall, for the less-experienced group members, the neural network tasks have been a tough start to the group exercises.

Keyword Spotting

For the keyword spotting task, we had split the task to three stages: Preprocessing, feature extraction and dynamic time warping.

For the preprocessing of the images, we first implemented binarization of the scanned pages where the use of local thresholding gave the best results. The available transcription file was converted to a dictionary (locations as keys, transcription as values); the outlines of the individual words were extracted from the available svg-files and used to excise the word images from the binarized page images. Files of word images were named using information in the

svg-file and the transcription dictionary created before. All images of individual words were then resized to the median width and height of all word images.

In the next step, image features (e.g. position of the highest or lowest black pixel, percentage of black pixels in centre part or lower third etc.) were extracted using column-wise scanning. Extracted features were normalized and stored in a feature matrix.

Last, dynamic time warping (DTW) was carried out to determine the distances between different words. A self-written DTW-function including Sakuro-Chiba-band turned out to slow to be useful, so we turned to an existing DTW package.

Overall, keyword spotting of our implementation was ok, but not splendid. Probably, it might have been beneficial to add more features, in particular, some comparing characteristics between, not just within the same column. Nevertheless, to our delight misclassified words at least often resembled the actual keyword to a good extent on visual inspection, reflecting the difficulties which also a human reader has at times with deciphering handwriting.

Overall, the keyword spotting task was probably the one which was most compatible with different levels of experience and certain group size (plus, we had improved cooperation and task sharing after the first task.)

Molecules Classification using graph matching

For the molecule classification task, the available gxl-files were converted to adjacency matrices using an existing package (xml.etree.ElementTree). The cost matrix for (inexact) bipartite graph matching was generated using an indel (insertion or deletion) cost of 2, and a substitution cost of 4 (= cost for one deletion plus one insertion).

To compare two molecules, we used the Kuhn-Munkres algorithm (aka Hungarian algorithm, published in 1955 by Harold Kuhn (Kuhn H., 1955), reviewed in 1957 by James Munkres (Munkres J., 1957), based on the earlier works of two Hungarian mathematicians, Dénes Kőnig and Jenő Egerváry (https://en.wikipedia.org/wiki/Hungarian_algorithm)). For efficient calculation of the cost, we made use of the linear_sum_assignment-function of the SciPy-package.
KNN-classification of the molecule distances was used to classify the individuals' compounds as either “active” or “inactive”. This approximation of the bipartite graph matching resulted in surprisingly good classification results (surprisingly, because not all information present in the molecule graph is taken into account like 3D structure, in principle just atoms and their specific bonds). Let’s see how it performs on the unknown test dataset.

We (and probably other groups as well) appreciated very much that the deadline for this task was extended by one week. Moreover, towards the end of the semester more and more assignments are due - it was pleasant that this was not the hardest task after all and for us as

bioinformaticians surprisingly successful experience by just applying the methods and theory from the lecture slides.

General thoughts

The group work was a good experience, in terms of improving the performance of our collaboration and the exercises were extremely useful to see how the theoretical problems have to be solved in practice. Large groups are maybe good to solve an exercise in a shorter time, but in smaller groups, the learning effect would be higher (But then it would be a problem, that we would have not enough time to solve the exercises). For the final results, it would be a little bit more efficient if we had the report formats in advance, which will prevent us from generating results in some unneeded format sometimes even not making sense but required time to develop.
In general, the support from the TAs was extremely helpful and motivating. We appreciate the work of the whole teaching team very much!

References

Harold W. Kuhn, "The Hungarian Method for the assignment problem", *Naval Research Logistics Quarterly*, **2**: 83–97, 1955.

https://en.wikipedia.org/wiki/Hungarian_algorithm

J. Munkres, "Algorithms for the Assignment and Transportation Problems", *Journal of the Society for Industrial and Applied Mathematics*, **5**(1):32–38, 1957 March.