Ramona Beck, Albana Jaha, Luca Althaus, Peter Allemann, Nathalie Froidevaux

PowerPuff Grills - Pattern Recognition FS2019 - Final Report

1. Group Organization

Channels:
- We used **WhatsApp** as communication platform to organize meetings and to update each other with the current progress and state of work.
- On **GitHub** we created some issues to assign tasks to the team members and worked on different branches to develop independently.
- We also had weekly **meetings** to plan, discuss the progress and the next steps.

Planning:
- For each task we discussed how we could split the work into subtasks and how we could assign them to the team members.
- We tried to estimate the effort for each subtask and their dependencies.
- Finally, we defined deadlines for each subtask to make sure that we will comply with the given submission date.

Execution:
- Depending on the subtask, we worked alone, in pairs or in a subgroup.

2. Tasks

2.1. Task 2

- **Our approach**
  We splitted the subtasks in the group and worked in parallel. For Task b), c) and d) we used the DeepDiva framework to train and test the networks.

- **What worked, what didn't work**
  - Some group members didn't manage to install or execute the DeepDiva framework at all.
  - In case of a running DeepDiva framework it further took a while to get warm with the handling. The documentation is short and clear though and in the end the framework was straightforward to use.
  - The visualization with the integrated Tensorboard was really cool, too.

a) SVM

- **Our approach**
  We implemented the SVM algorithm for three different kernels: linear, polynomial and RBF, using cross-validation and parameter tuning.
  To get the best parameter combination, we created arrays for possible C-, gamma-, degree-values and used GridSearchCV from the sklearn library.

- **What worked, what didn't work**
  The computation of the best parameter combination never finished, not even on the cluster.
  We could test the execution for the single kernels and with given parameters, but still, the training took ages.

b) MLP

- **Our approach**
  To build the architecture of the MLP model, we got the inspiration from the already existing models in DeepDiva.
  To get the best accuracy, we tried different value combinations for
  - learning rate (0.001, 0.003, 0.01, 0.03, 0.1)
  - hidden layers / number of neurons (10, 30, 50, 100, 500)
  - epochs (5, 10, 20)

- **What worked, what didn't work**
  We could observe that the bigger the hidden layer size and the epochs, the higher is the accuracy.
  We also observed that if the learning rate is small (e.g. 0.001) we have to train longer.

c) CNN

- **Our approach**
  In order to complete the provided CNN implementation we
  1. investigated the shape (*width and height*) and number of *input-channels* of the images with the appropriate Python-function.
  2. calculated the number of *output-channels* and the *kernel size* that it fits to the predefined classifier-layer with the formula given in the lecture.
  3. provided the final linear layer with the *output size*, which was given by the number of classes in the MNIST dataset.
  Finally, we trained the CNN with different learning rates and different amount of epochs to find the best accuracy on the validation set.

d) Permutated MNIST

- **Our approach**
  - MLP: we simply used the best performing parameters from task 2b).
- **What worked, what didn't work**
  After performing Task 2b) and 2c) it was straightforward to perform this task, also because we already knew the DeepDiva framework and we had some knowledge about the best performing parameters (MLP).

2.2 Task 3

- **Our approach**
  In order to fulfill this task we performed the following:
  1. We iterate over all keywords.
  2. For every keyword, we get *one* corresponding train word.

3. We calculate the distance between this train word to every validation word by applying the “Distance Time Warping” algorithm, with the help of the “fastdtw” implementation (https://pypi.org/project/fastdtw/). For the DTW we extracted the following features: lower contour, upper contour, the number of black pixels between lower and upper contour, the number of black pixels between lower and upper contour, the number of black pixels and the number of black/white transitions.
4. We sort the distances and select the most similar words according to a given threshold.
5. Finally, we calculate the confusion matrix to get the accuracy and recall.

- **What worked, what didn’t work**
  - We had some problems with the keyword spotting task since the task description is unclear (for example, we did not really understand what the output of the task should be).
  - It was much harder to organize this task, since there were many dependencies and working in parallel was not really possible, in contrast to task 2.
  - We didn’t implement the Precision-Recall-Curve to test the performance of our code.

2.3 Task 4 - Molecules

- **Our approach**
  We implemented the molecule Task using linear_sum_assignment from scipy.optimize which directly implements the Hungarian algorithm.
  1. We first translated all gxl files into python objects containing all the needed information, since we use them multiple times.
  2. We created the Dirac matrix according the literature and the scipy-documentation for the algorithm explained how to handle the results to calculate/approximate GED.
  3. Then we simply iterated through both sets and calculated the distance matrix using knn (like in exercise 1).
  4. We tried to optimize the parameters Ce, Cn and k. With Ce=1, Cn=1 and k=3 we achieved good results and more testing didn’t improve the results further.

- **What worked, what didn’t work**
  - It took a while to understand how to use gxl-files (but now it’s nice to know).
  - With the scipy-documentation the usage of the Hungarian algorithm and the calculation of the GED was straightforward.

3. Conclusion / Feedback

- Make knowledge about programming mandatory.
- Maybe give cluster access from the beginning? It really helped us.
- Task 2: This was a very cool task, since it was easy to split the workload in the group and you could work in parallel. And we learned a lot about (neural) networks and the impact of the different parameters.
- Task 3: Really unclear task description, hard to split because no work can be done in parallel.