Pattern Recognition - Final Report

JungleSpeed

Group Organization

GitHub link : `https://github.com/MCapallera/PatternRecognition_JungleSpeed`
We split the group tasks the following way :

MNIST / Permutated MNIST		Keyword Spotting	Molecule	Documentation and Report
SVM	Melania, Simon	Melania, Simon	Marine, Quentin	Melania, Simon, Marine, Quentin
CNN	Marine, Quentin / Melania, Simon			
MLP	Marine, Quentin / Melania, Simon			

Two students are from Bern (Melania, Simon) and two from Fribourg (Marine, Quentin). In the exercise sessions we discussed who is going to solve which part of the exercises. During the week, Whats-App was used for communication. For each task, one group was in charge of the implementation of the task and the other group was in charge of the revision and the validation. We discussed about the results we got at the end of each task but also if something was wrong or missing.

MNIST SVM task

For optimize the prediction of the Numbers in the MNIST dataset we used the SVM and the GridSearchCV from the scipy package. Because the run-time for the plain data was more than some hours long, we transformed the features with the hog transform into hog-features.This reduced the features space around 95% (from 784 to 36 features). Also we transformed the hog-features to the range from 0 to 1, as suggested in a blog [1]. We run the grid search with following kernels and corresponding parameter configurations : lienar kernel with $C : [0.1, 1, 5]$, rbf kernel with $C = [0.1, 1, 5]$ and $\gamma = [0.001, 0.01, 0.05]$

MNIST CNN task

The goal of this exercise was to train a CNN with only one convolutional layer and experiment with different parameters. We first had to find the right values for the input channels, output channels and the kernel size of the convolutional layer. The input size was 28x28 px. The stride was set to 3 with no padding. According to : o = ((i + 2p - k)/s) + 1, with o the output size. We found that the output size had to be 8 and the output channels 24, since the input dimension of the linear layer was set to 1536 (8*8*24). Therefore the kernel size (k) was set to 7. We run the algorithm in order to optimize some hyperparameters. The optimal parameters for this model were : learning rate = 0.1, epochs = 35

MNIST MLP task

The goal of this exercise was to train an MLP with one hidden layer and experiment with different parameters. We implemented the MLP using Keras. We first optimize the number of neurons in the hidden layer, then the learning rate and finally the number of epochs. We figured out that there was no need to run it for more than 50 epochs. According to the previous results, we performed random initialization to refine our parameters and choose the model performing the best. All the steps can be found in the GitHub project [2].

The optimal parameters for this model were : number of neurons = 62, learning rate = 0.1, epochs = 40

1. `https://medium.com/@basu369victor/handwritten-digits-recognition-d3d383431845`
2. PatternRecognition_JungleSpeed/reports/JungleSpeed_Task2b_V1.pdf

CNN and MLP with the permuted MNIST dataset

Another task was to test the CNN and MLP implemented before on another dataset : the MNIST dataset with permuted pixels. The MLP showed to perform the same way with a similar accuracy for the permuted MNIST. The permuted pixel does not seem to have an influence on the classification for this network. For CNN, the permuted MNIST has a slightly lower accuracy than with the normal MNIST dataset. Since the CNN is taking into account the spatial features in the images, the fact that the digits have been randomly permuted have made it harder for the CNN to classify the digits.

Keyword Spotting task

Task description

Goal : find similar word images in the manuscript.
Provided Data :

- ground-truth : Contains the transciption.txt and locations - transcription.txt : Contains the transcription of all words (on a character level) of the whole dataset ; locations : Contains bounding boxes for all words in the svg-format.
- Images : Contains the original images in jpg-format.
- Task : files for train and validation names and keywords to spot.

Implementation

In our solution we provide the user a configurable, job based solution which allows to configure the parameters and order of the tasks (e.g. binarize the image, crop...). This was done in order to find the optimal preparing steps/tasks with the optimal parameters. Additionally it allows to prepare and run different configurations easily on the cluster.

The prepossessing was done in the following way :

1. The pictures were cropped according to the SVG paths provided in the ground truth.
2. Then images were binarized by the Yen Threshold-Algorithm.
3. Because the provided SVG paths aligned not exactly to the shape of the word, we computed a difference-image between the binarized and a background image (all pixel are set to the background color). The bounding box from this result where then used to crop the images from the step one. These pictures were cropped in the width in the way that maximally 3 white pixels were left on each side .
4. Then the height was normalized over all pictures

For feature extraction the hog transformation was used, as suggested in several keywords spotting papers. The dynamic time warp algorithm with Sakoe Chiba band was used for the comparison of the feature vectors of two pictures. Unfortunately we got a somewhat low precision around (42%) and recall (72%).

For each transcription a cluster was built and an optimal threshold was calculated. The threshold was chosen the following way : We calculated for each point in the cluster the distances to every other point within the same cluster. The point that minimizes the sum of this distances was then chosen to be the cluster mean (midpoint). Then the distances of every point to the midpoint was calculated and the threshold is adjusted such that the acceptance rate of elements that truly belong to the set is maximized and the acceptance rate of elements that do not belong to the set is minimized.

Results

Problems

We first had the problem that our accuracy and precision were really low. We then recognized that we did not properly define the centers of our clusters. This probably led to too high thresholds. First we were not sure which parameters we should use in the hog-transformation. Finally we found a paper which suggested to use 12 orientations - this clearly improved our results.

orientations	recall (%)	precision (%)	arccuracy (%)
20	71.73	42.55	92.19
16	73.06	39.64	90.91
12	72.08	42.38	93.06

Table 1 – Results from the KWS with different orientations we tested

Molecules task

Task description

Goal : Classify the molecules of the validation set using KNN with the approximate Graph Edit Distance (GED). Provided data : **Graph xml** (gxl files) with nodes labeled with their chemical symbol and unlabeled undirected edges **-** **Labels for molecules** (active and inactive) with their IDs, split into a training and a validation set

Implementation

The first step of this task was to extract information from gxl files in Python. This is done in *app/Molecules/-getData.py.* To do that, we used the *xml.etree.ElementTree* library to get nodes and edges for each molecule and we added them into graphs that can be created in python by using the library *networkx.* A dictionary with all graphs is returned.

The second step was to compute the approximate Graph Edit Distance (GED) between pairs of molecules with bipartite graph matching. Then, we use the optimal Edit Path for each pair of molecules and perform a classification using the k-NN algorithm.

From the dictionary of all graphs, we first separated graphs into a training set and validation set using the ids of graphs in the text files (train.txt and valid.txt). Labels of training set and validation set have been also saved in a dictionary. Then, we are calculating the k-closest neighbours from the training set, for each graph of the validation set. The k-nearest neighbours are calculated based on the approximate GED. This is done in the library that we imported in the algorithm package. For each pair of graphs, the cost matrix is built in *abstract_graph_edit_distance.py.* Then, we apply the Hungarian Algorithm through the function *linear_sum_assignment()*, which is already implemented in the *scipy.optimize package.* The algorithm returns the graph edit distance between the two graphs. The distance is normalized on the size of the two graphs. The k-NN algorithm keeps the k IDs and graphs of molecules that have the smallest normalized distance. Then, a prediction is done for each molecule of the test set based on the labels of the k-nearest neighbours computed before.

Output and Results

For each molecule, the prediction (active or inactive) is exported in a text file with the format *ID, class prediction.* This file is exported in *results/MoleculesClassification/output_k-NN.txt.* At the same time, we are computing the accuracy. If the prediction is matching with the label of the molecule found in the valid.txt file, a correct answer is counted. At the end, the total accuracy of the algorithm is computed and printed in the console. The results of our algorithm on the proposed training and validation sets for different values of k can be found in Table 2.

Table 2 – Accuracy of the kNN classification for different values of k

Number of neighbours	3	5	7	10	15
Accuracy (%)	98.4	**99.2**	**99.2**	**99.2**	98.4

Conclusion

There exercises were interesting to solve. We got a deeper understanding and insight in the topics treated during the lectures. The group was in the way challenging that we didn't see each other except during the lecture and exercise sessions. However WhatsApp was enough to organize the group work.