UNIVERSITÄT HEIDELBERG
Zukunft. Seit 1386.
IWR Interdisciplinary Center for Scientific Computing

Deep Vision
Summer Semester 19
Prof. Björn Ommer
ommer@uni-heidelberg.de

Exercise sheet 2: Linear Classifier and Support Vector Machine

Due on *06/05/2019*, 11:00.

Biagio Brattoli (biagio.brattoli@iwr.uni-heidelberg.de).

Important

1. **EMAIL**: check which e-mail address is assigned to your Moodle account and check it frequently! All notifications regarding the course will be sent via Moodle.

2. **PROCESSING TIME**: The exercise sheets are usually uploaded 1 week before the solution needs to be submitted online via moodle. Due to bank holidays the processing time might be extended. The deadline is always indicated at the beginning of the exercise sheet.

3. **SUBMISSION**: ONE ZIP-File and ONE-PDF. The names of both files should contain your surname.

General Information:

All the exercises must be completed using Python. The proposed solution for the exercises will be compiled in Python 3. You may use standard Python libraries or Anaconda (open source distribution of Python) to complete your exercises.
As for Deep Learning there are a lot of good libraries. We suggest you to use PyTorch[1], since it is easy to work with. Alternatively, you can use Caffe[2], Tensorflow[3] or Theano[4]. We would not recommend, however, to start with the last mentioned libraries if you are not already familiar with them or you definitley want or need to learn them.
If you have any problems or questions about the exercise, you are welcome to use the dedicated forum on Moodle (most likely others share the same question). For technical issues about the course (for example if, for some reason, you cannot upload the solution on moodle) you can write an email to the responsible of the exercise (indicated at the beginning of the exercise sheet). About the grading system, keep in mind that you can get full points for one question even if the solution is incorrect. However, the submitted solution must show a necessary effort to complete the task, so please upload the answer to the question even if it is not

[1] `http://pytorch.org/`
[2] `http://caffe.berkeleyvision.org/`
[3] `https://www.tensorflow.org/`
[4] `http://deeplearning.net/software/theano/`

completed. The solution submitted on moodle must contain a PDF file containing the code and the results (numbers, plots, images, ...), plus you need to upload the necessary code to reproduce the results in a compilable version ('.py' files).

Exercise 2:
For this exercise, a template code is provided in *exercise2.py*. The script provides the functions needed to handle the data. Addditionality, in the same file, we offer a template for solving each task, where the student needs to implement the #TODO sections. Suggested libraries are also imported at the beginning of the file. However, you are not restricted by the template, which is ment to be a simply guide, so you can change any part if needed.
Before starting, you need to download the CIFAR10 dataset, like in the first exercise, and set the path in the variable "cifar10_dir" (look at exercise1 for more details).

Task 1: Linear Classifier with Least Square (3P)
During the lecture, you have learned about linear classifier, i.e. a function $f(x|w,b) = sign(x \cdot w^T + b)$. The parameters (w, b) can be optimized in many ways.
For this task you need to implement a linear classifier and optimize the parameter using a simple least-square minimization algorithm. For the optimization you can use the function *scipy.optimize.leastsq* or any other framework you like.
After solving the binary classification Airplain VS Frog based on color histogram, report the train and test accuracy and plot the optimized parameters using meaningful xticks labels.
In sintesy

1. Implement linear classifier optimized by least-square
2. Train classifier on binary task and report accuracy
3. Plot learned parameters

Task 2: Linear Support Vector Machine (1P)
For the second task, solve the same binary problems as in the task1, but using a linear SVM. We suggest to use *sklearn.svm.LinearSVC* since it is very fast, however you can use any existing implementation. Again, report the accuracy and plot the learned parameters.
Finally, compare the results obtained from the two optimization process, Least-Square and SVM, and comment on performances and learned parameters.

Tip: What you will probably experience is that Least-Square outperforms SVM for this task, due to the simple feature representation used in the exercise. In case more representative features are used, SVM generally outperforms Least-Square. If you want to play around with different features, exchange the function "get_features" with

```
from skimage.feature import hog
def get_features(img):
        return hog(img)
```

However, this should not be part of the uploaded solution.

Task 3: Non-Linear SVM and Cross-Validation **(2P)**

During last lecture, SVM with a non-linear kernel was introduced, which can solve more complex problem respect to the linear counterpart by mapping the data in an higher dimensional space. Solve the binary problem using a Non-Linear SVM, implemented by *sklearn.svm.SVC*.

Given the high capacity of non-linear classifier, it is fundamental to tune the hyper-parameters. Before train the classifier on the full training set, use cross-validation to find good hyper-parameter. In case of the SVM, you need to select: kernel type, soft-margin C and gamma.

In sintesy

1. Implement cross-validation.
2. Find the best hyper-parameters for the SVM on the binary task.
3. Train SVM on full traininig set and report accuracies.
4. Comment on the result.

Task 4: Multi-Class Linear Classifier **(2P)**

In task1 you have implemented a binary linear classifier. A binary classifier can be used to solve multi-class problems by one-vs-all approach:

1. Given a dataset $D = [(x_1, y_1), (x_2, y_2), ...]$ of pair $x_i \in \mathbb{R}^{F \times 1}$ image features and y_i class label.
2. For a class c, assign x_i to the positive class when $y_i = c$, to the negative class when $y_i \neq c$.

3. Optimize the weights $w_c \in \mathbb{R}^{1\times F}$ and bias b_c for the classifier of class c.
4. Repeat [2,3] for all C classes.
5. Stack all weights w_i with $i = 1, ..., C$ into a matrix $W \in \mathbb{R}^{C\times F}$, where F is the features size, and biases $b \in \mathbb{R}^{C\times 1}$.
6. During infearence, predict the class by $\hat{y} = argmax(W \cdot x + b)$

To solve this task, train the multi-class classifier and report the test accuracy.

Task 5: Multi-Class Linear SVM **(1P)**
Solve the same multi-class problems as in the previous task, but using *sklearn.svm.LinearSVC*, which already implements one-vs-all approach by default, and report the accuracy.

Task 6: Classification by Logistic Regression with Pytorch **(1P)**
In *exercise2.py* you will find a solution to the multi-class problem using a simple linear classifier implemented in Pytorch (for this exercise, the cpu version is sufficient). Your task is to try to understand the code and answer the following questions:

- What does *nn.Linear* do?
- What is a *criterion*?
- What happen when *loss.backward(); optimizer.step()*?
- Why do you think we need to reduce the learning rate after 1000 epochs?

Then you need to modify the code in order to plot the loss over time.

Tip: Since many topics have not been presented yet during the lectures, we do not expect you to answer correctly. The goal of this task is to make yourself familiar with some standard concept in deep learning (like criterion, optimizer, epoch and loss) and get in touch with some Pytorch basics.