Exercise 1a

Deadline: 20.11.2020, 16:00

In this exercise, you will familiarize yourself with Python, Jupyter, Numpy and Matplotlib and verify some theoretical findings from the lecture via Monte Carlo simulation.

Regulations

Please implement your solutions in form of *Jupyter notebooks* (`*.ipynb` files) which can mix executable code, figures and text in a single file. They are created in the browser app JupyterLab or its legacy version Jupyter Notebook.

Create a Jupyter notebook `monte-carlo.ipynb` for your solution and export the notebook to HTML as `monte-carlo.html`. Zip both files into a single archive with naming convention (sorted alphabetically by last names):

`lastname1-firstname1_lastname2-firstname2_ex01a.zip`

or (if you work in a team of three)

`lastname1-firstname1_lastname2-firstname2_lastname3-firstname3_ex01a.zip`

and upload this file to Moodle before the given deadline. Remember that you have to reach 50% of the homework points to be admitted to the final mini-research project.

Preliminaries (not graded)

Create a Python environment containing the required packages using the `conda` package manager[1] by executing the following commands on the command line:

```
conda create --name ml_homework python    # create a virtual environment
conda activate ml_homework                # activate it (set paths etc.)
conda install scikit-learn matplotlib     # install packages into active environment
python                                    # run python
```

This brings up Python's interactive prompt. When everything got installed correctly, the following Python commands should load the respective modules without error:

```
import numpy        # matrices and multi-dimensional arrays, linear algebra
import sklearn      # machine learning
import matplotlib   # plotting
```

To install and run Jupyter, execute the following on the command line (note: this only works when environment `ml_homework` is active, see the `activate` command above)

```
conda install -c conda-forge jupyterlab    # install JupyterLab (only needed once)
jupyter lab                                # opens JupyterLab in your web browser
```

If your are not familiar with Python, Jupyter, and/or the numerics package `numpy`, check out our tutorial on Wednesday, 2pm or work through these tutorials:

- `http://www.datacamp.com/community/tutorials/tutorial-jupyter-notebook`
- `https://www.physi.uni-heidelberg.de/Einrichtungen/AP/Python.php` (in German)
- `https://cs231n.github.io/python-numpy-tutorial/`

[1] You can download `conda` (specifically, its basic variant `miniconda`) from `https://conda.io/miniconda.html`. It is also part of the Anaconda software distribution, which you may already have installed.

1 Monte-Carlo Simulation

In the lecture, we considered the following toy problem: The feature variable $X \in [0,1]$ is real-valued and 1-dimensional, and the response $Y \in \{0,1\}$ is discrete with two classes. The prior probabilities and likelihoods are given by

$$\begin{aligned} p(Y=0) = p(Y=1) &= \tfrac{1}{2} \\ p(X=x|Y=0) &= 2-2x \\ p(X=x|Y=1) &= 2x \end{aligned}$$

We also derived theoretical error rates of the Bayes and nearest neighbor classifiers for this problem. Monte Carlo simulation is a powerful method to verify the correctness of theoretical results experimentally.

1.1 Data Creation and Visualization (7 points)

Since the given model is generative, one can create data using a random number generator. Specifically, one first samples an instance label Y according to the prior probabilities, and then uses the corresponding likelihood to sample the feature X. If no predefined random generator for the desired likelihood is available (as is the case here), uniformly distributed samples from a standard random number generator can be transformed to the desired distribution by means of "inverse transform sampling" (see `https://en.wikipedia.org/wiki/Inverse_transform_sampling`).

Work out the required transformation formulas for our likelihoods and show your derivation in a Markdown cell. Then implement a function

```python
def create_data(N):
    return ... # data X and labels Y
```

that returns the X-values and corresponding Y-labels for N data instances. Use the module `numpy.random` to generate random numbers. Check that the data have the correct distribution with `matplotlib` (see `https://matplotlib.org/gallery/statistics/hist.html` for a demo).

1.2 Classification by Thresholding (5 points)

In the lecture, we defined a classification rule deciding according to a threshold $x_t \in [0,1]$:

- Rule A (threshold classifier): $\hat{Y} = f_A(X; x_t) = \begin{cases} 0 & \text{if } X < x_t \\ 1 & \text{if } X \geq x_t \end{cases}$
- Rule B (threshold anti-classifier): $\hat{Y} = f_B(X; x_t) = \begin{cases} 1 & \text{if } X < x_t \\ 0 & \text{if } X \geq x_t \end{cases}$

  (it always predicts the opposite of rule A)

The corresponding error rates are:

$$p(\text{error}|A; x_t) = \frac{1}{4} + \left(x_t - \frac{1}{2}\right)^2$$

$$p(\text{error}|B; x_t) = \frac{3}{4} - \left(x_t - \frac{1}{2}\right)^2 = 1 - p(\text{error}|A; x_t).$$

Confirm experimentally for $x_t \in \{0.2, 0.5, 0.6\}$ that the predicted error rates are correct: Verify that the minimum overall error of 25% is achieved when the threshold $x_t = 0.5$ (optimal Bayes classifier). Repeat each test with 10 different test datasets of the same size M and compute mean and standard deviation of the error. Use test set sizes $M \in \{10, 100, 1000, 10000\}$. How does the error standard deviation decrease with increasing M?

1.3 Baseline Classifiers (2 points)

We now compare the above results to two rules that entirely ignore the features:

- Rule C (guessing): $\hat{Y} = f_C(X) = \begin{cases} 0 & \text{with probability } \frac{1}{2} \\ 1 & \text{otherwise} \end{cases}$
- Rule D (constant): $\hat{Y} = f_D(X) = 1$ (it always predicts class 1)

Both result in an error rate of 1/2. Confirm this like in the previous exercise: Plot the error and its standard deviation as a function of test set sizes $M \in \{10, 100, 1000, 10000\}$ for both new rules.

1.4 Nearest Neighbor Classification (6 points)

Implement the nearest neighbour classifier for our toy problem. Make sure that it can handle training sets of arbitrary sizes.

Sample a training set of size $N = 2$. Make sure it contains one instance of either class, so your `create_data` from the first exercise will not work out of the box. Determine the error rate of the nearest neighbour classifier on a sufficiently large test set. Repeat this with 100 different training sets (all of size $N = 2$) and compute the average error on the same test set. Verify that this average error is around 35%.

Repeat the experiment with training sets of size $N = 100$ and report the average error.