Exercise 1

Deadline: 31.10.2018, 15:00

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

Regulations

Please implement your solutions in form of *jupyter notebooks* (*.ipynb files). Jupyter is a very elegant Python GUI, where executable code, figures and text can be embedded next to each other in nicely formatted notebook files.

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_exercise1.zip

or (if you work in a team of three)

lastname1-firstname1_lastname2-firstname2_lastname3-firstname3_exercise01.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
source activate ml_homework              # activate it (set paths etc.) on Linux/Mac
# on Windows, 'source' is not needed, you just call
#      activate ml_homework
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 jupyter    # install jupyter (only needed once)
jupyter notebook         # opens jupyter in your web browser
```

If your are not familiar with Python, Jupyter, and/or the numerics package numpy, 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 (as density functions f) are given by

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

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

```
features, labels = create_data(N)
```

that returns vectors containing the X-values and corresponding Y-labels for N data instances. Use module `numpy.random` to generate random numbers. Check that the data have the correct distribution with `matplotlib`'s histogram function (see `https://matplotlib.org/1.2.1/examples/pylab_examples/histogram_demo.html` for a demo).

1.2 Classification by Thresholding (7 points)

In the lecture, we defined two classification rules depending on 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)

and claimed that their average error rates are

$$\begin{aligned} p(\text{error}|A, x_t) &= \frac{1}{4} + \left(x_t - \frac{1}{2}\right)^2 \\ p(\text{error}|B, x_t) &= \frac{3}{4} - \left(x_t - \frac{1}{2}\right)^2 = 1 - p(\text{error}|A, x_t) \end{aligned}$$

Moreover, we can define two rules that ignore the features, resulting in an error rate of 1/2

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

Confirm experimentally for $x_t \in \{0.2, 0.5, 0.6\}$ that the predicted error rates are correct. In particular, verify that the minimum error of 25% is achieved when rule A is applied with 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 Nearest Neighbor Classification (6 points)

Implement the nearest neighbor classifier for our toy problem such that it can handle training sets of arbitrary sizes.

Create a training set of size $N = 2$, which contains one instance of either class. Use rejection sampling to achieve this: reject the second sample when it has the same class as the first and sample again. Determine the error rate of the nearest neighbor 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 35%.

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