zeroth_exercise

November 9, 2020

1 Zeroth Exercise

1.0.1 1. 1 Register as a Group in Ilias

For submission please form a group of three students. You have to assign to a group in Ilias. You can use the "Homework Group Finding Forum" (https://ilias.studium.kit.edu/ilias.php?ref_id=1307493&cmd=showThreads&cmdClass=ilrepositorygui&cmdNo to team up. Although you will submit the homework as a group, to avoid confusions we ask all students to submit the homework in Ilias. Please make sure that every group member submits the same homework version.

1.0.2 1.2 Introduction

This is a test exercise to get familiar with the way the exercises are presented in the Lecture "Machine Learning - Grundverfahren".

The exercise sheets comes in form of Jupyter notebooks, which consists of different cells. These can be used for markdowns (like this cell) or Python code. Each cell can be run separately to present the text properly or to execute the code. It is also possible to use LaTeX equations inside the markdown environment. For more information about how to install and start Jupyter, visit https://jupyter-notebook.readthedocs.io/en/stable/

The given Jupyter notebook should be filled out by where it is marked. **The exercises have to be solved and submitted as a group of three students**.

There are either pen and paper questions to answer or programming tasks to code. You should answer the programming tasks inside this notebook. The theoretical questions can be answered here as well using the Markdown cells. It is also possible to answer them by pen and paper and scan the solution as a pdf.

For the submission of the exercise please follow these steps:

1. Make sure that every cell is executed and the output is printed.
2. Create a PdF of the Jupyter notebook via *File* → … → *PDF via LaTeX (.pdf)* or *File* → *Print Preview* → *Use your favorit PDF printing program*
3. Zip your created PdF file and your original notebook, i.e. the .ipynb file, as well as your separate pen and paper solution if existent together.
4. Rename your zip file with the following naming convention: group_y_uxxxx_uxxxx_uxxxx where y is your group number, uxxxx is the kit user from each group member/

5. Upload the zip file to Ilias. Please make sure that every group member uploads the group submission.

For this exercise, please try to fill in the code where stated. It demonstrates some basic operations we will need for this course. Then, submit this exercise following the instructions above. This submission is only for getting familiar with the submission procedure. It will not be graded.

1.0.3 Exercise 1: Matrices and Vectors in Python

As mentioned, we will use Python for the coding exercise. We will use version 3.6. Additionally, for this and all following exercise we will use NumPy, one of the most fundamental python libraries, designed to efficiently create and operate on multi-dimensional arrays (, i.e., vectors, matrices and higher order tensors)

If you are new to NumPy we refer to https://numpy.org/devdocs/user/quickstart.html .

1.1 Create Matrices and Vectors in Python We first create a matrix

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 2 & 6 \\ 7 & 3 & 8 \end{pmatrix}$$

and two vectors

$$v = \begin{pmatrix} 3 \\ 1 \\ 2 \end{pmatrix}, \; w = \begin{pmatrix} 1 \\ 1 \\ 2 \end{pmatrix}$$

[6]:
```
import numpy as np

# Let us create a matrix A and vectors v and w:
A = np.array([[1, 2, 3],
              [4 ,2, 6],
              [7, 3, 8]])
v = np.array([3, 1, 2])
w = np.array([1, 1, 2])
print("Matrix A:")
print(A)
print("Vector v:")
print(v)
print("Vector w:")
print(w)
```

```
Matrix A:
[[1 2 3]
 [4 2 6]
 [7 3 8]]
```

```
Vector v:
[3 1 2]
Vector w:
[1 1 2]
```

1.2 Basic Operations with Vectors and Matrices Implement the v + w and 2 · A. Print out your results and verify that they are correct.

[7]:
```
# We can perform general element-wise operations:
elem_wise_sum = np.add(v, w) # do the element-wise sum v+w here
print(elem_wise_sum)
print("----------")
elem_wise_mult = np.multiply(2, A) # do the element-wise multiplication 2*A here
print(elem_wise_mult)

# The element-wise sum of v and w is correct, as:
# [3+1, 1+1, 2+2] = [4, 2, 4]    | Math, simple Addition
#
# The element-wise multiplication of A with the scalar 2 is correct, as:
# [[2*1, 2*2, 2*3],      [[ 2,  4,  6],
#  [2*4, 2*2, 2*6],    =  [ 8,  4, 12],    | Math, simple Multiplication
#  [2*7, 2*3, 2*8]]       [14,  6, 16]]
```

```
[4 2 4]
----------
[[ 2  4  6]
 [ 8  4 12]
 [14  6 16]]
```

Do the matrix vector product A · v. Print out your result and verify that the result is correct.

[8]:
```
# To perform a matrix multiplication, use either np.dot or just the "@" symbol:
matrix_vector_prod = np.dot(A, v) # do the matrix vector product A*v here
print(matrix_vector_prod)

# This is correct as the Matrix A has as many columns as the vector v rows
# and the multiplication is computed as follows:
#   [1*3 + 2*1 + 3*2, 4*3 + 2*1 + 6*2, 7*3 + 3*1 + 8*2]
# = [ 4  +  2  +  6 , 12  +  2  + 12 , 21  +  3  + 16 ]    | Math, Simple
 ↪Multiplication
# = [      11       ,       26       ,       40       ]    | Math, Simple
 ↪Addition
```

```
[11 26 40]
```

Do the matrix matrix product A · A. Print out your result and verify that the result is correct.

[9]:
```
# Two matrices can be multiplied in the same fashion:
matrix_matrix_prod = np.dot(A, A) # do the matrix matrix product A*A here
print(matrix_matrix_prod)

# This is correct as the Matrix A has as many columns as the Matrix A rows
# and the multiplication is computed as follows:
#   [[1*1 + 2*4 + 3*7, 1*2 + 2*2 + 3*3, 1*3 + 2*6 + 3*8],
#    [4*1 + 2*4 + 6*7, 4*2 + 2*2 + 6*3, 4*3 + 2*6 + 6*8],
#    [7*1 + 3*4 + 8*7, 7*2 + 3*2 + 8*3, 7*3 + 3*6 + 8*8]]
# = [[ 1  +  8  + 21 ,  2  +  4  +  9 ,  3  + 12  + 24 ],
#    [ 4  +  8  + 42 ,  8  +  4  + 18 , 12  + 12  + 48 ],    | Math, Simple␣
 ↪Multiplication
#    [ 7  + 12  + 56 , 14  +  6  + 24 , 21  + 18  + 64 ]]
# = [[ 30, 15,  39],
#    [ 54, 30,  72],    | Math, Simple Addition
#    [ 75, 44, 103]]
```

```
[[ 30  15  39]
 [ 54  30  72]
 [ 75  44 103]]
```

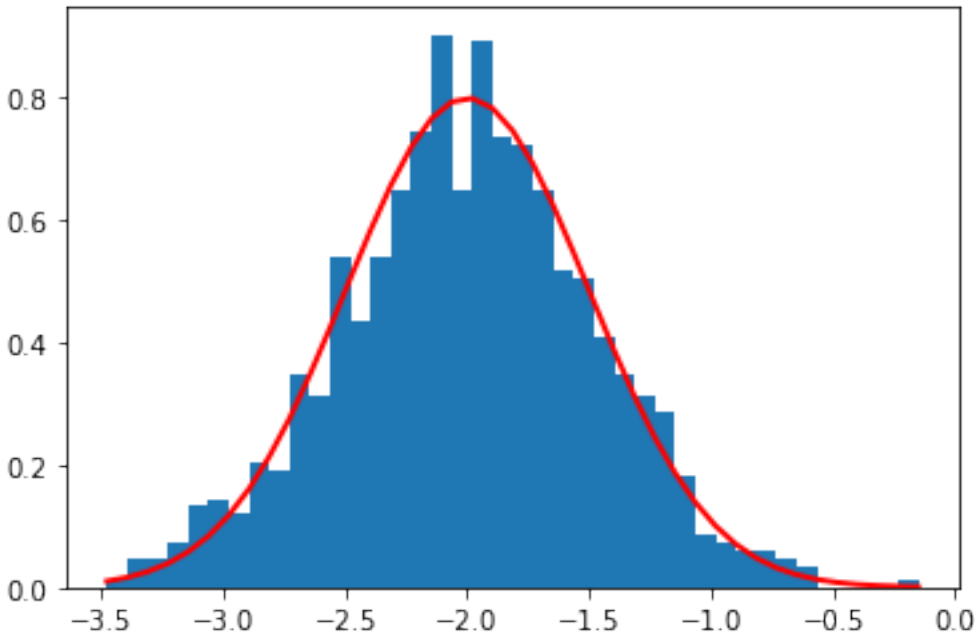
1.0.4 Exercise 2: Sampling from a Gaussian

Sample 1000 samples from a Gaussian distribution with mean $\mu = -2$ and standard deviation $\sigma = 0.5$.

We also visualize the samples. For this we use matplotlib, a python package for plotting

[10]:
```
# for jupyter notebooks you need the following line of jupyter magic command␣
 ↪(that's literally the name)
# to display matplotlib figures inside the notebook. Make sure it comes before␣
 ↪the actual import!
%matplotlib inline
#actually import matplotlib
import matplotlib.pyplot as plt

# In Machine Learning, it is often necessary to sample from a probability␣
 ↪distribution.
# To sample from a Gaussian distribution (https://en.wikipedia.org/wiki/
 ↪Normal_distribution):

mu, sigma = -2, 0.5 # mean and standard deviation
samples = np.random.normal(mu, sigma, 1000) # do the sampling from the normal␣
 ↪distribution here
# print(samples[0:10])

# plot the histogram of the samples
```

```
count, bins, ignored = plt.hist(samples, 40, density=True)

# plot the density function
plt.plot(bins, 1/(sigma * np.sqrt(2 * np.pi)) * np.exp( - (bins - mu)**2 / (2 *␣
 ↪sigma**2) ),  linewidth=2, color='r')

# show the plot
plt.show()
```



1.0.5 Exercise 3: Solving a Linear Equation

You are given the linear equation

$$Ax = v.$$

We want to find the vector x. Assume that the matrix A and the vector v are given as in Exercise 1.1. Find the solution for x. Hint: Use the function 'linalg.solve' from numpy - this should always be preferred over direct matrix inversion.

[11]:
```
# To solve the linear equation system A x = v, you may use the np.linalg.solve␣
 ↪function. In other words we can
# use np.linalg.solve for inverting any matrix which does not stand alone.

x = np.linalg.solve(A, v) # solve for x here

print("Solution: x =", x)
```

```
print("Test: A@x =", A@x, "is the same as v =", v, ":",
      np.all(np.ndarray.astype(A@x, int) == v), "!")
```

```
Solution: x = [-0.08333333  2.41666667 -0.58333333]
Test: A@x = [3. 1. 2.] is the same as v = [3 1 2] : True !
```

1.1 Closing Remarks

NumPy provides easy and efficient ways to deal with vectors, matrices and tensors. Linear algebra functionality like the function introduced above, together with more advanced techniques such as broadcasting (https://numpy.org/doc/stable/user/basics.broadcasting.html), are the key to efficient programming in Python. If you want to do any practical Machine Learning you need to familiarize yourself with those techniques. We will use and introduce more of this in the following exercises but also expect you to use it for your submission. So use them whenever possible and, most importantly, avoid unnecessary for loops.