

General constraints for code submissions Please adhere to these rules to make our and your life easier! We will deduct points if your solution does not fulfill the following:

- If not stated otherwise, we will use Python 3.9.
- If not stated otherwise, we expect a Python script, which we will invoke exactly as stated on the exercise sheet.
- Your solution exactly returns the required output (neither less nor more) – you can implement a `--verbose` option to increase the verbosity level for developing.
- Add comments and docstrings, so we can understand your solution.
- (If applicable) The `README` describes how to install requirements or provides additional information.
- (If applicable) Add required additional packages to `requirements.txt`. Explain in your `README` what this package does, why you use that package and provide a link to its documentation or GitHub page.
- (If applicable) All prepared unittests have to pass.
- (If applicable) You can (and sometimes have to) reuse code from previous exercises.

The automated machine learning methods you will learn about in this course help you to improve your skill and knowledge for applying machine learning in practice. The goal of this first exercise is to set up teams and learn about git and the workflow for future exercises. Having learned about different ways to empirically

evaluate the performances of algorithms and AutoML systems, in this exercise you will now implement some of these techniques. Add your code creating plots and outputting statistics to `main.py` (callable as `python main.py`).

We provide a simple Makefile which you can use to install all packages listed in your requirements file (`make install`).

For all questions including a statistical test, we consider the p-value threshold to be 0.05.

1. McNemar Test [3 points]

Two models are trained to classify images of cats and dogs. The result is stored in `MCTestData.csv` with $n = 500$ images. The function `load_data_MNTest()` loads the data as an $n \times 3$ *numpy array*, where the first column represents the ground truth. The 2nd and the 3rd columns represent the output from model 1 and 2 respectively.

Implement a *McNemar Test* to determine whether the two models perform equally well on the dataset. In your solution, state what is H_0, H_1 and return χ^2 for this evaluation. As this exercise will not be automatically graded, please include your answer in an attached PDF file, and upload that PDF to your repository.

2. Two-Matched Samples t-Test [3 points]

`TMStTestData.csv` contains *error* values of two algorithms on $n = 419$ datasets, the function `load_data_TMStTest()` loads the data as an $n \times 2$ *numpy array*.

Implement a *Two-Matched-Samples t-Test* to determine whether the two algorithms perform equally well on the dataset and return the test statistic t value for this evaluation.

3. Friedman Test [3 points]

`FTestData.csv` contains *error* values of $k = 5$ algorithms on $n = 15$ datasets, the function `load_data_FTest()` loads the data as an $n \times k$ *numpy matrix* `Err`, where `Errij` represents the error of the j th algorithm on the i th dataset.

Implement a *Friedman Test* to determine if all algorithms are equivalent in their performance and return χ_F^2 for this evaluation. If this hypothesis is not rejected, you can skip the next question.

4. Post-hoc Nemenyi Test [3 points]

Having found that all the algorithms are not ranked equally, now we need to utilize the *Post-hoc Nemenyi Test* to find the best-performing algorithm.

Compute the test statistic for all the algorithms pairs $\{j_1, j_2\}$. The results should be stored in an upper triangular matrix **Q**, where $Q_{m,n}$ is the q value between the algorithms j_m and j_n

5. Boxplots [2 points]

Create a boxplot¹ for error value of the algorithms which have the best and the worst average ranks stored in *FTestData.csv*. This exercise will not be automatically graded. To let us quickly grade your exercises please add your solutions to the PDF file. Further we expect all plots to have axes labels and a legend.

6. Code Style [1 point]

On every exercise sheet we will also make use of *pycodestyle*² to adhere to a common python standard. Your code will be automatically evaluated on every push and you will be informed if the test fails. To check it locally, first run *pip install pycodestyle* and then run *pycodestyle -max-line-length=120 src/* to check your source file folder. Alternatively run *make check*

7. Feedback [1 bonus points]

Please give us feedback by filling out the feedback.md file.

- Major Problems?
- Helpful?
- Duration (hours)? For this, please follow the instructions in the feedback.md file.
- Other feedback

¹using e.g. https://matplotlib.org/api/_as_gen/matplotlib.pyplot.boxplot.html

²former pep8