Heinrich-Heine-Universität Düsseldorf
Computational Cell Biology
Alexander Kroll

Applications of Transformer Networks in Bio- and Cheminformatics

Worksheet 5

Submission Deadline:	27. May 2024, 11:59 pm
Discussion of solutions:	28. May or 4. June 2024, 4:30 - 6:00 pm

---

Submission Instructions:

- Upload a Jupyter notebook with your solutions to the exercises that you solved on your own PC. Upload the Python scripts and log files for any code executed on the HPC.
- If you are submitting multiple files, zip them together.
- Submit your solutions by uploading the Jupyter notebook to https://uni-duesseldorf.sciebo.de/s/hCt1rTP23EeWmUC. The uploaded file should have the following filename: "lastname_studentID_worksheet5".

Excercise 5.1 *Predicting K_M using small molecule representations* (40 Points)

The Michaelis constant K_M of an enzyme-substrate pair describes the affinity of an enzyme for its substrate. Formally, K_M is defined as the substrate concentration at which the enzyme operates at half its maximal speed. Although the K_M of an enzyme depends on both the enzyme and the substrates, the structure of the substrate already contains a lot of information about the optimal K_M of the enzyme-substrate pair. In this exercise, we will use information about the substrate to predict the K_M value of an enzyme-substrate pair. For this task, you will need to download .csv files in *data/worksheet5*.

(a) Compute extended-connectivity fingerprints (ECFPs), RDKit fingerprints, MACCS keys, and molecular descriptors for each SMILES string in the downloaded K_M datasets:

- To compute the molecular descriptors, use all 208 descriptors from the *rdkit.Chem* module *Descriptors*. If the resulting arrays contain values greater than 10^6, replace them with zeros.
- To compute ECFPs, you can use the function *GetMorganFingerprintAsBitVect* from the rdkit.Chem module *AllChem*. Set the radius to 2 and nBits to 1024.

(b) For each of the four numerical small molecule representations computed in (a), perform a hyperparameter optimization of a gradient boosting model among the following parameters with the following upper and lower bounds:

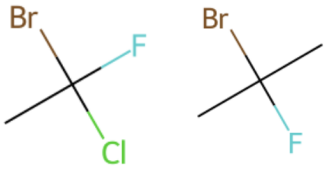
- Number of trees ("n_estimators"): Lower bound: 30, Upper bound: 500,
- Maximum tree depth: ("max_depth"): Lower bound: 2, Upper bound: 12,
- Learning rate: ("learning_rate"): Lower bound: 0.01, Upper bound: 0.5
- Min. child weight ("min_child_weight"): Lower bound: 1, Upper bound: 15,
- Lambda ("reg_lambda"): Lower bound: 0, Upper bound: 1,
- Alpha ("reg_alpha"): Lower bound: 0, Upper bound: 1.

Search for the best set of hyperparameters on the validation set by iterating over at least 50 different combinations of hyperparameters using the Python package *hyperopt*. Search for the set that leads to the highest coefficient of determination R^2 on the validation set.

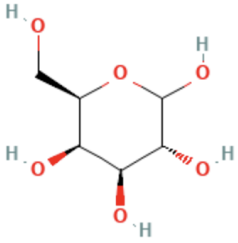
(c) For each small molecule representation, select the set of hyperparameters that yielded the best R^2 on the validation set in (b). Train a gradient boosting model with this set of hyperparameters on the training set and evaluate it on the test set, i.e., calculate MSE and R^2 for the test set.

Excercise 5.2 *Small molecule representations* (20 Points)

(a) For both of the following molecular graphs below, answer the question: Does this molecule contain a chiral center? Explain your answer.



(b) For the molecule shown below as a molecular graph, state the number of each atom type present in the molecule and provide a SMILES string representing the molecule (including its stereochemistry).



Excercise 5.3 *SMILES string tokenization* (40 Points)

If we want to use SMILES strings as input for Transformer networks, we need to tokenize SMILES strings. In this exercise, you will create your own tokenization scheme suitable for representing small molecule SMILES strings in a format that could be processed by a transformer network.

(a) Create a Python dictionary that defines your tokenization for SMILES strings. This dictionary should map each unique token in your tokenization scheme to a unique integer. Your tokenization should aim for a balance between granularity (capturing enough chemical detail) and efficiency (not too fragmented, i.e., not all tokens should consist of only 1 character). Your tokenization scheme only needs to successfully address patterns and atoms present in the following SMILES strings:

- C[C@H](O)C
- C([C@@H](CO)O)O
- C[C@H](O)C(=O)O
- CC(=O)O
- C(=O)O
- C/C=C\O
- CO[C@H](O)C(=O)O
- C1COC(=O)[O@H]1
- [O-]C(=O)C
- c1ccccc1

(b) Explain your choice of tokenization scheme in (a).

(c) Define a Python function that takes a SMILES string as input and returns a list of integers representing the tokenized version of the string according to your dictionary. Use your function to provide the tokenization for the SMILES strings listed in (a).

Excercise 5.4 *Use of LLMs* (0 Points)

State for which exercise you have used LLMs (large language models) such as ChatGPT or GitHub Copilot. State which tools you have used and for which steps. This answer does not influence how many points you receive for your submission.