MolPipeline : A python package for processing molecules with RDKit in scikit-learn

Jochen Sieg,[†,‡] Christian W. Feldmann,[†,‡] Jennifer Hemmerich,[†] Conrad Stork,[†] Frederik Sandfort,[†] Philipp Eiden,[†] and Miriam Mathea[*,†]

†*BASF SE, Ludwigshafen. Germany*

‡*Equal Contribution*

E-mail: miriam.mathea@basf.com

Abstract

The open-source package scikit-learn provides various machine learning algorithms and data processing tools, including the Pipeline class, which allows users to prepend custom data transformation steps to the machine learning model. We introduce the MolPipeline package, which extends this concept to chemoinformatics by wrapping default functionalities of RDKit, such as reading and writing SMILES strings or calculating molecular descriptors from a molecule object. We aimed to build an easy-to-use Python package to create completely automated end-to-end pipelines that scale to large data sets. Particular emphasis was put on handling erroneous instances, where resolution would require manual intervention in default pipelines. In addition, we included common cheminformatics tasks, like scaffold splits and molecular standardization, natively in the pipeline framework and adaptable for the needs of various projects.

Introduction

Molecular property prediction is a fundamental challenge for the chemical sciences, drug discovery and one of the oldest tasks in cheminformatics.[1,2] Recent developments push the field from traditional QSAR models[3] to more expressive deep learning methodologies.[4] Simultaneously, standard approaches, like tree-based methods with molecular fingerprints, are competitive and represent strong baselines,[4,5] especially in low-data domains common in real-world scenarios.[4,6] Therefore, the most effective method for a new task often depends on the specific circumstances, requiring the evaluation of different approaches. For practitioners, this means building models regularly and manually for different datasets, tasks, and target properties.

The typical steps for building a new machine learning model for property prediction consist of first processing the molecules in a suitable representation, like a feature vector, and then applying machine learning-specific processing and algorithms to these features. Two widely employed packages are RDKit[7] and scikit-learn.[8] RDKit can process molecules, including reading different input formats, filtering out unwanted compounds, standardizing molecules, and computing various molecular features. scikit-learn can use the molecular feature matrix generated with RDKit to perform feature selection or reduction, set up cross-validation experiments, and train and select prediction models. While these are standard tasks, they can require manual work that can become repetitive, time-consuming, and error-prone. Manually handling these steps does not scale for large-scale evaluations, requiring robust automation of the process. Simultaneously, different projects might require different molecular standardization, custom data splits and featurization, and heterogeneous molecular data from various sources demands simple and flexible adaptation to the project's needs. Therefore, an easy-to-use, robust, flexible, and scalable end-to-end software framework is necessary.

Several packages addressing these problems have been introduced over the years.[9–14] For example, PREFER[11] automates the model and feature selection process for classical machine

learning and deep learning. Datamol[15] and MolFeat[12] provide a light wrapper for RDKit and a wide range of classical and modern molecular representations. Scikit-chem[14] and Scikit-Mol[13] connect RDKit-functionality with scikit-learn. However, most frameworks for integrating molecular feature representations into machine learning models do not include or allow custom handling of regular cheminformatics tasks, like molecular standardization, data splits or combining individual features. Those that allow for both still need to address invalid inputs or molecules failing in the preprocessing steps in an automated way. Therefore, it is challenging to achieve end-to-end automation while allowing processing steps to change flexibly depending on the project's requirements.

We propose MolPipeline, a Python package unifying cheminformatics tasks for molecular machine learning in a single scalable pipeline implementation. Like preceding tools, MolPipeline incorporates the standard open-source libraries RDKit[7] for cheminformatics and scikit-learn[8] for machine learning. With MolPipeline , we aim to enable flexible automatic end-to-end learning from a molecular data set to a trained machine learning model ready for productive deployment. The key features of MolPipeline are

- Automated end-to-end processing from molecule data sets to deployable machine learning models.
- Scalable parallel processing and low memory usage through instance-based processing.
- Standard pipeline building blocks for flexibly building custom pipelines for various cheminformatics tasks.
- Consistent error handling for tracking, logging, and replacing failed instances. For example, a SMILES string that could not be parsed correctly.
- Integrated serialization for reusing pipelines and tracking them in version control.

With the release of MolPipeline, we provide a tool covering a broad spectrum of use cases, such as, custom molecular standardization (e.g., salts removal), molecule-based and

data-driven clustering (e.g. Murcko scaffolds), and molecular encoding. In addition, different encodings can be combined to a single feature vector, e.g. feature vectors of fingerprints and physicochemical properties. The final pipeline is compatible with scikit-learn's API and for hyperparameter optimization, model selection, and can be serialized for deployment or as a template for future projects. Compared to previous tools, a key feature in our work is the consistent handling of processing errors. As in real-world data sets, it is not uncommon to come across SMILES strings that cannot be successfully parsed. Automatic logging, filtering, or replacing these samples is mandatory to avoid manual intervention, which can become cumbersome for large data regimes and is infeasible for end-to-end productive deployment in automatic workflows.

Implementation

Pipeline concept

MolPipeline extends the pipeline concept from scikit-learn[8,16] to processing and transforming molecular input data using RDKit. These pipelines allow to perform a sequence of user defined transformations on the input data and finalize the procedure with an optional machine learning model for predictions. As the individual transformations remain accessible, corresponding parameters can be optimized with typical tools for hyperparameter optimization, like scikit-learn's GridSearchCV.[8]

In MolPipeline, we provide new pipeline elements implementing standard cheminformatics tasks using RDKit while complying with scikit-learn's pipeline API. These pipeline elements are building blocks with well-defined input and output. Therefore, they can be flexibly arranged to create new pipelines for specific needs, like custom molecular standardization. The pipeline elements in MolPipeline are grouped into the modules `any2mol`, `mol2mol`, `mol2any` and custom `estimators`.

Any2Mol Pipeline elements in the `any2mol` module transform a given molecular representation or input format to the corresponding RDKit molecule object. For example, the `SmilesToMol` element creates RDKit molecule objects from SMILES. We also implemented an `AutoToMol` element that automatically determines the input format for reading molecules from multiple input formats, like SMILES, SDF, or binary, without manually specifying the input format.

Mol2mol The `mol2mol` module provides pipeline elements having a molecule as input and output, for example, pipeline elements for molecular standardization, like `TautomerCanonicalizer`, `MetalDisconnector`, `SaltRemover`, `FragmentDeduplicator` and more. Another pipeline element in the module is, for example, the `MurckoScaffold` element to extract a molecule's Murcko scaffold.

Mol2any The pipeline elements in the `mol2any` module transform RDKit molecule objects into various representations, ranging from molecular file formats to feature vectors. On the one hand, elements like `MolToSmiles`, `MolToInchi`, `MolToBinary` convert the molecule in SMILES, Inchi, and RDKit's binary format, respectively. On the other hand, molecules can be featurized with `MolToMorganFP` and `MolToRDKitPhysChem`. Further, the feature vectors generated with individual pipeline elements can be scaled separately and concatenated with `MolToConcatenatedVector`.

Custom estimators In the `estimator` module, we created custom estimators complying with the Scitkit-learn API. These estimators implement common cheminformatics tasks, like clustering molecules. For example, the `MurckoScaffoldClustering` estimator performs a clustering based on Murcko scaffolds. Like scikit-learn's clustering estimators, the `MurckoScaffoldClustering` is usable in pipelines and can be parallelized and parameterized, for example, to compute the generic scaffold or employ different strategies to handle linear molecules during the clustering.

Error handling In MolPipeline, we use a dedicated object called `InvalidInstance` to mask failed instances. When an instance, like a molecule, fails a pipeline step, we substitute it with an `InvalidInstance`, which can be detected, skipped, and replaced automatically. We provide the two pipeline elements `ErrorFilter` and `FilterReinserter` to remove such `InvalidInstances` from the processing and replace them with fill-values, respectively. These features enable the automatic processing of large data sets without manually fixing non-processable instances.

Instance-based processing Whenever possible, MolPipeline processes instances independently, achieving an effective parallelization and reduction memory usage. Before execution, we preprocess the computation graph of the MolPipeline and split it into parts where instances can be processed concurrently and parts that need synchronization. For example, the standardization procedure of molecules might comprise multiple subsequent steps, like salt removal, neutralizing charges, and selecting a canonical tautomer. This subsequent standardization process can be handled independently for each molecule, which allows for effective parallelization. In contrast, synchronization is required when applying column-wise standardization, like z-scaling, to a feature matrix, which can only be applied when the entire feature column is constructed. MolPipeline automatically detects when instances can be processed independently and when synchronization is required. In addition, instance-based processing reduces memory usage because memory-intensive intermediate results are only kept in memory as long as needed.

Serialization A configured pipeline can be stored in common configuration files (e.g., JSON) and pickle (e.g., through joblib as scikit-learn uses). Serialization based on configuration files allows configured pipelines to be saved and reused, for example, the same pipeline for a different data set. With pickle-based serialization, fitted models can be stored, e.g., for deployment on a server.

Documentation and code quality We provide a series of Jupyter notebooks to exemplify the usage of MolPipeline at `https://github.com/basf/MolPipeline/tree/main/notebooks`. The MolPipeline project has also been developed using continuous integration/development (CI/CD) best practices to ensure the library's high quality and documentation. The quality of the code is also continuously tested with unit tests and statically monitored with linters.

Examples and results

Basic usage

Figure 1 shows the basic usage to construct a pipeline for building a prediction model from a SMILES data set. The pipeline converts the SMILES into molecules using the `Auto2Mol` pipeline element. Then, the molecules are standardized with the `ElementFilter` removing molecules with uncommon chemical elements that are often hard to process in subsequent steps. In addition, the `SaltRemover` strips commonly used counterions, such as Na^+ or Cl^- from molecules. Next, Morgan fingerprints are computed from the standardized molecules with `MolToMorgan`, which are then used as the input feature vectors to the `RandomForestRegressor` from scikit-learn. The number of cores for parallel execution can be specified with the `n_jobs` argument. Finally, the created pipeline can be fitted with a data set and used for predictions as scikit-learn's estimator interface defines.

```python
# set up pipeline
pipeline = Pipeline(
    [
        ("auto2mol", AutoToMol()),                                  # reading molecules
        ("element_filter", ElementFilter()),                        # standardization
        ("salt_remover", SaltRemover()),                            # standardization
        ("morgan2_2048", MolToMorganFP(n_bits=2048, radius=2)),     # fingerprints and featurization
        ("RandomForestRegressor", RandomForestRegressor()),         # machine learning model
    ],
    n_jobs=16,
)

# fit the pipeline
pipeline.fit(X=["CCCCCC", "c1ccccc1"], y=[0.2, 0.4])
# make predictions from SMILES strings
pipeline.predict(["CCC"])
# >>> array([0.29])
```

Figure 1: Code example for using MolPipeline for molecular machine learning.

The example in Figure 2 shows how to use MolPipeline's `MurckoScaffoldClustering` estimator, a custom estimator for scaffold clustering of molecules implementing scikit-learn's interface. On construction, `MurckoScaffoldClustering` can be parameterized to use Murcko scaffolds or generic scaffolds (all atoms are considered carbons and all bonds are considered single bonds) using the `make_generic` option. In addition, the handling of linear molecules, which would not yield a scaffold, can be specified with the parameter `linear_molecules_strategy`. The here shown option `"own_cluster"` assignes all linear molecules to a separate cluster. Finally, the clustering estimator can cluster molecules using their scaffolds through the scikit-learn API using functions like `fit_predict`.

```python
scaffold_smiles = [
    "Nc1ccccc1",
    "Cc1cc(Oc2nccc(CCC)c2)ccc1",
    "c1ccccc1",
]
linear_smiles = ["CC", "CCC", "CCCN"]

# run the scaffold clustering
scaffold_clustering = MurckoScaffoldClustering(
    make_generic=False, linear_molecules_strategy="own_cluster", n_jobs=16
)
scaffold_clustering.fit_predict(scaffold_smiles + linear_smiles)
# output: array([1., 0., 1., 2., 2., 2.])
```

Figure 2: Code example of MolPipeline's custom estimators. Shows the estimator for clustering by Murcko scaffolds.

Hyperparamter optimization example

The more advanced example illustrated in Figure 3 shows how MolPipeline can be used for hyperparameter optimization in conjunction with scikit-learn's classes to identify the best-performing machine learning model and feature vector on the BBBP data set.[17] We combine MolPipeline with scikit-learn's `GroupShuffleSplit` and `GridSearchCV` and evaluate Random Forest (RF), K-NearestNeighbors (KNN) and Logistic Regression (LR) with variations of the Morgan fingerprint.

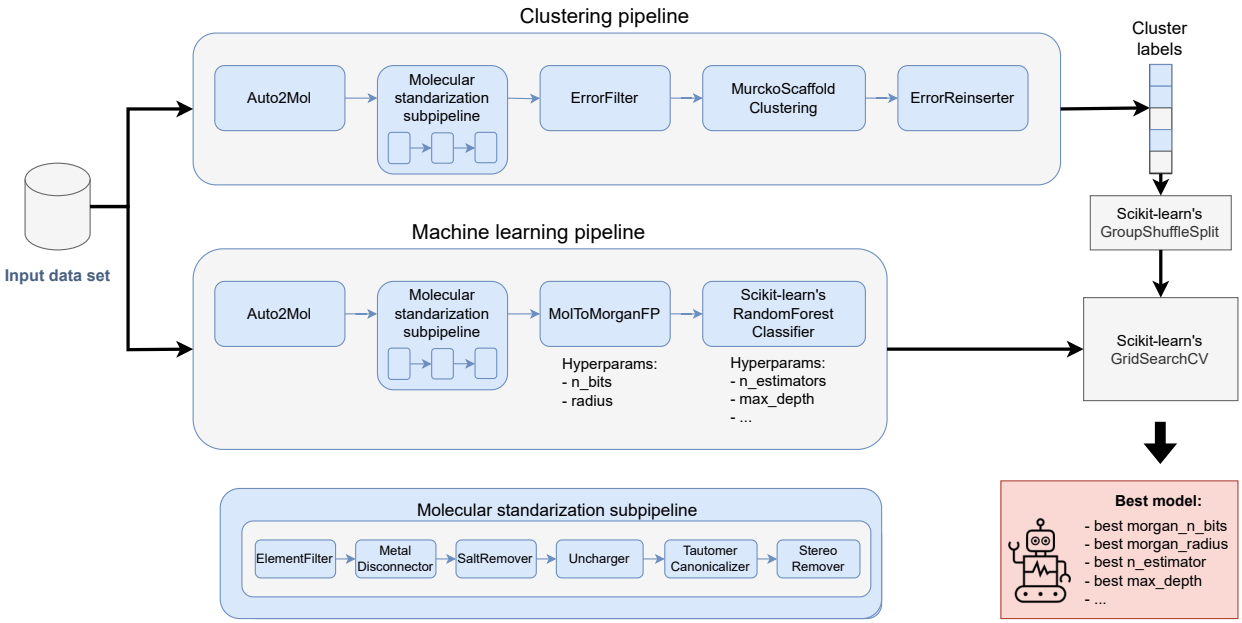

Figure 3: Schematic overview of two MolPipelines to perform hyperparameter optimization with scikit-learn's `RandomForestClassifier`, Morgan fingerprints, and scaffold splits with scikit-learn's `GridSearchCV` class. Note that besides the `RandomForestClassifier` hyperparameters, the parameters of the Morgan fingerprints are included during optimization, e.g., the number of bits and the fingerprint radius.

Molecular standardization For molecular standardisation, we compiled the commonly used procedure into a subpipeline, as illustrated in Figure 3. The `ElementFilter` removes molecules containing chemical elements not in a predefined list. The pipeline elements `MetalDisconnector` and `SaltRemover` remove metals and metal ions from a given molecule. The `Uncharger` standardized the molecule's charge and the `TautomerCanonicalizer` computes a canonical tautomer. Finally, the `StereoRemover` removes stereo information from the molecule.

Clustering pipeline and splitting The clustering pipeline (see Figure 3) performs scaffold clustering. It starts by reading the molecules with the `Auto2Mol` pipeline element, then standardizes them using the subpipeline. Molecules that failed the reading or standardization procedure are removed before clustering using the `ErrorFilter` element. The clustering of the valid molecules is computed with the `MurckoScafflodClustering` estimator, which

groups molecules into clusters using their Murcko scaffold. Lastly, the resulting list of cluster labels is processed by the `ErrorReinserter` element that places fill values (e.g. Numpy's NaN) at the positions of failed molecules. The resulting label list can be mapped element-wise to the input molecule list through this reinsertion step. The clustering generated 1033 clusters on the BBBP data set. Their size distribution is illustrated in Figure 4.

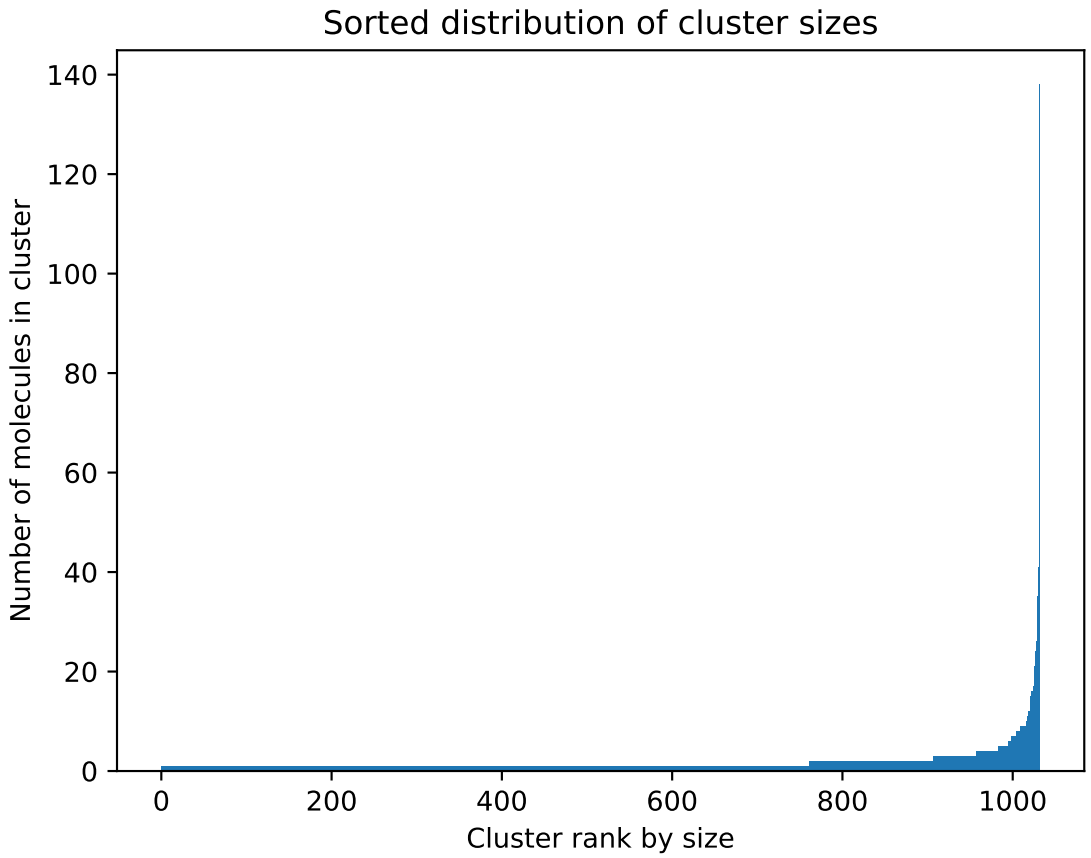

Figure 4: Distribution of scaffold cluster sizes on the 2050 molecules of the BBBP data set.

The computed cluster labels are then used to group the molecules of the BBBP data set with scikit-learn's `GroupShuffleSplit` into 80% training, 10% validation, and 10% test sets, keeping molecules with the same scaffold in the same fold. The training and validation sets are used for hyperparameter tuning, and the test set is used to evaluate the performance of the selected pipeline with the best fingerprint and model hyperparameters.

Machine learning pipeline The machine learning pipeline defines a prediction model, including its processing from the raw molecular input (see Figure 3). This pipeline starts with the same reading and molecular standardization subpipeline as the clustering pipeline. Morgan fingerprints are computed using the `MolToMorganFP` pipeline element from the standardized molecules. The feature vectors are input to scikit-learns machine learning estimators, like `RandomForestRegressor`. For the sake of simplicity, invalid molecules detected during the assingment of the cluster labels are removed from the data set used for the ML experiments. To account for erroneous molecules occurring during the training and prediction phase, the `ErrorFilter` and `ErrorReinsert` elements can be added to the ML pipeline as well.

Grid search results We tried out different hyperparameter sets for Random Forest, K-NearestNeighbors, Logistic Regression and Morgan fingerprints. The results of these experiments are illustrated in Figure 5, which shows the distributions of the different hyperparameter sets for each model on the validation set. The best ROC-AUC on the hold-out test set is 0.95 using RF Morgan fingerprints with radius 1 and 2048 bits. The second best model is KNN with a 0.91 ROC-AUC and the third LR with a 0.89 ROC-AUC. The hyperparameter tuning took about 4.5 minutes on a laptop using 16 cores.

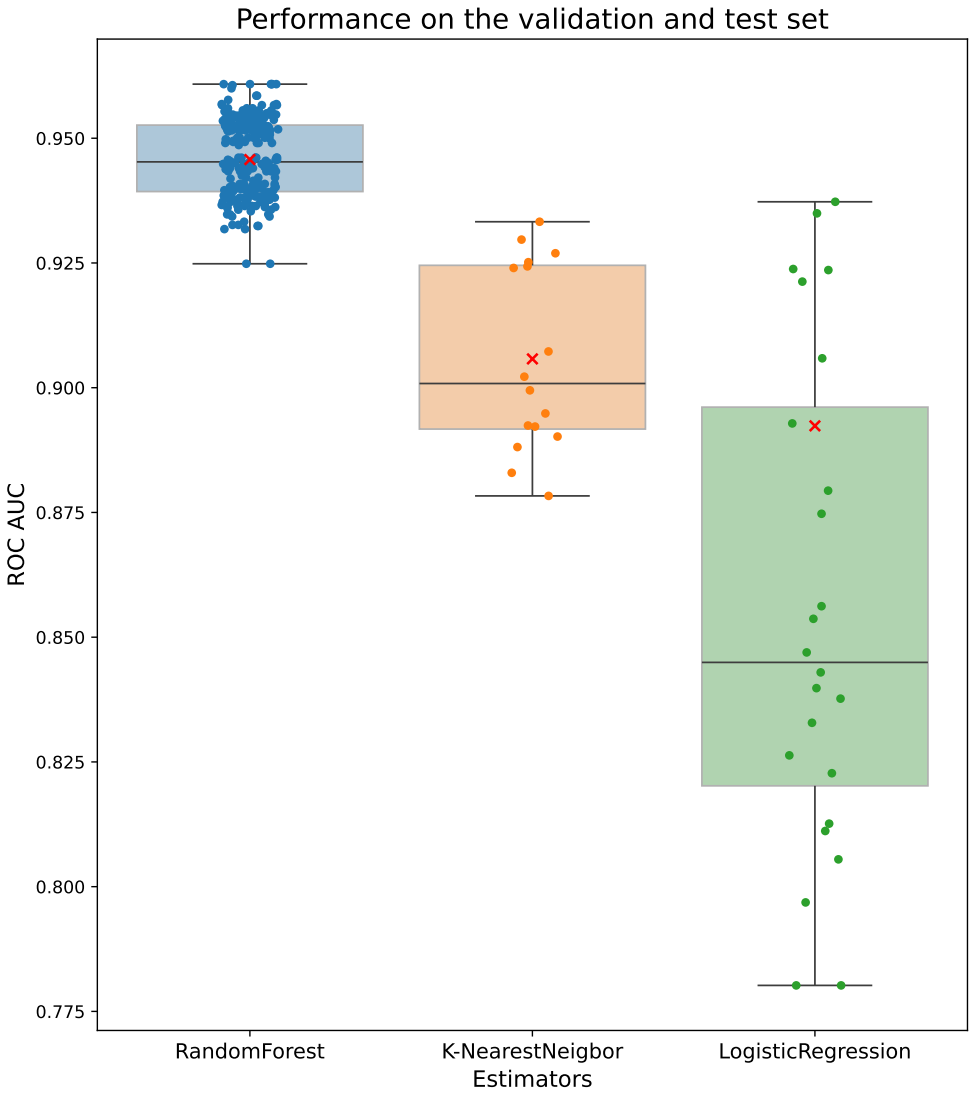

Figure 5: Distribution of ROC-AUC performance on the validation set by different hyperparameter combinations. The red x indicates the ROC-AUC on the hold-out test set of the best models selected on the validation set.

Analyzing failed molecules While processing the BBBP data set, MolPipeline logged eleven SMILES that failed the reading stage in the `Auto2Mol` element. The failed SMILES contain uncharged tetravalent nitrogen atoms, and RDKit correctly fails to process them since tetravalent nitrogen atoms should be charged. The molecules correspond to the eleven molecules described by Patrick Walters.[18] Through MolPipeline's automatic error handling, the pipelines could execute without manual interventions. However, further manual inves-

tigation of the failed cases can help understand the data better and decide whether failed cases can be fixed either manually or by adding additional pipeline elements.

Conclusion

We propose MolPipeline, a package that aims to automate the model-building process while allowing it to flexibly adjust to the needs of different projects. By combining RDKit and scikit-learn, their functionality is unified. With the MolPipeline package, we aim to make this combination accessible to practitioners for their daily cheminformatic tasks, model building, and handling the ever-increasing data size while avoiding unnecessary re-implementation. By leveraging the pipeline concept from scikit-learn and adding instance-based processing and consistent error handling, we hope to provide a robust implementation suited for various molecular machine learning tasks. We plan to maintain and add further functionality to the MolPipeline in the future, including clustering algorithms and modern deep learning estimators. MolPipeline is available at `https://github.com/basf/MolPipeline`.

Software & Data Availability Statement

MolPipeline is openly available at `https://github.com/basf/MolPipeline` under MIT license.

References

(1) Yang, K.; Swanson, K.; Jin, W.; Coley, C.; Eiden, P.; Gao, H.; Guzman-Perez, A.; Hopper, T.; Kelley, B.; Mathea, M.; others Analyzing learned molecular representations for property prediction. *Journal of chemical information and modeling* **2019**, *59*, 3370–3388.

(2) Bender, A.; Schneider, N.; Segler, M.; Patrick Walters, W.; Engkvist, O.; Rodrigues, T.

Evaluation guidelines for machine learning tools in the chemical sciences. *Nature Reviews Chemistry* **2022**, *6*, 428–442.

(3) Tropsha, A. Best practices for QSAR model development, validation, and exploitation. *Molecular informatics* **2010**, *29*, 476–488.

(4) Deng, J.; Yang, Z.; Wang, H.; Ojima, I.; Samaras, D.; Wang, F. A systematic study of key elements underlying molecular property prediction. *Nature Communications* **2023**, *14*, 6395.

(5) Jiang, D.; Wu, Z.; Hsieh, C.-Y.; Chen, G.; Liao, B.; Wang, Z.; Shen, C.; Cao, D.; Wu, J.; Hou, T. Could graph neural networks learn better molecular representation for drug discovery? A comparison study of descriptor-based and graph-based models. *Journal of cheminformatics* **2021**, *13*, 1–23.

(6) Stanley, M.; Bronskill, J. F.; Maziarz, K.; Misztela, H.; Lanini, J.; Segler, M.; Schneider, N.; Brockschmidt, M. Fs-mol: A few-shot learning dataset of molecules. Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2). 2021.

(7) RDKit: Open-source cheminformatics. https://www.rdkit.org.

(8) Pedregosa, F. et al. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* **2011**, *12*, 2825–2830.

(9) Dixon, S. L.; Duan, J.; Smith, E.; Von Bargen, C. D.; Sherman, W.; Repasky, M. P. AutoQSAR: an automated machine learning tool for best-practice quantitative structure–activity relationship modeling. *Future medicinal chemistry* **2016**, *8*, 1825–1839.

(10) Kausar, S.; Falcao, A. O. An automated framework for QSAR model building. *Journal of cheminformatics* **2018**, *10*, 1–23.

(11) Lanini, J.; Santarossa, G.; Sirockin, F.; Lewis, R.; Fechner, N.; Misztela, H.; Lewis, S.; Maziarz, K.; Stanley, M.; Segler, M.; others PREFER: A New Predictive Modeling Framework for Molecular Discovery. *Journal of Chemical Information and Modeling* **2023**, *63*, 4497–4504.

(12) Noutahi, E.; Wognum, C.; Mary, H.; Hounwanou, H.; Kovary, K. M.; Gilmour, D.; thibaultvarin r; Burns, J.; St-Laurent, J.; t; DomInvivo; Maheshkar, S.; rbyrne momatx datamol-io/molfeat: 0.9.4. 2023; `https://doi.org/10.5281/zenodo.8373019`.

(13) Bjerrum, E. J.; Bachorz, R. A.; Bitton, A.; Choung, O.-h.; Chen, Y.; Esposito, C.; Ha, S. V.; Poehlmann, A. Scikit-Mol brings cheminformatics to Scikit-Learn. **2023**,

(14) Lewis, R. P. I.; Delicious., M. G. scikit-chem: pre-alpha. 2016; `https://doi.org/10.5281/zenodo.60137`.

(15) Mary, H. et al. datamol-io/datamol: 0.12.3. 2024; `https://doi.org/10.5281/zenodo.10535844`.

(16) Buitinck, L.; Louppe, G.; Blondel, M.; Pedregosa, F.; Mueller, A.; Grisel, O.; Niculae, V.; Prettenhofer, P.; Gramfort, A.; Grobler, J.; Layton, R.; VanderPlas, J.; Joly, A.; Holt, B.; Varoquaux, G. API design for machine learning software: experiences from the scikit-learn project. ECML PKDD Workshop: Languages for Data Mining and Machine Learning. 2013; pp 108–122.

(17) Martins, I. F.; Teixeira, A. L.; Pinheiro, L.; Falcao, A. O. A Bayesian approach to in silico blood-brain barrier penetration modeling. *Journal of chemical information and modeling* **2012**, *52*, 1686–1697.

(18) We Need Better Benchmarks for Machine Learning in Drug Discovery. https://practicalcheminformatics.blogspot.com/2023/08/we-need-better-benchmarks-for-machine.html, Accessed: 2024-03-28.