Reactions & python packages

Practical Programming in Chemistry

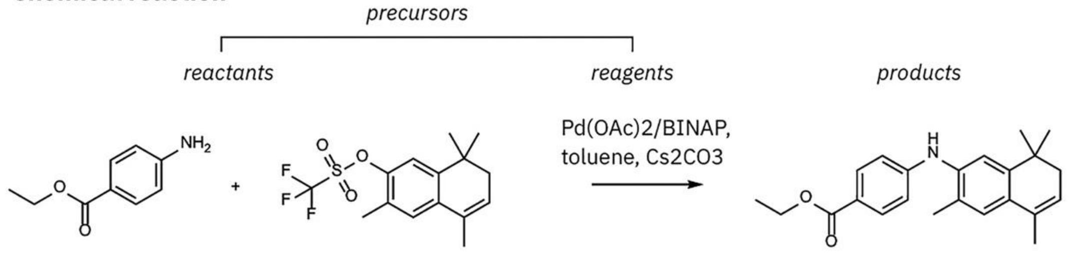
Prof. Philippe Schwaller

This lecture

- A bit more on chemical reactions
- Python packages

Recap from last time

Chemical reaction



Meta data

reaction class - 1.3.4
Buchwald-Hartwig amination

reaction yield - 80%

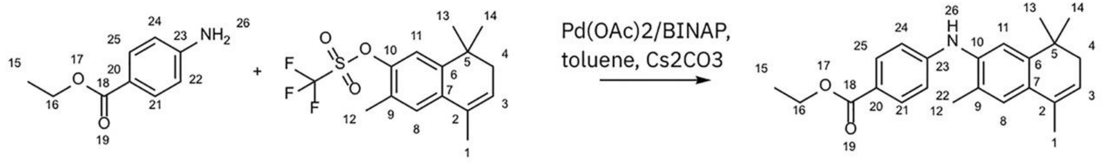
experimental procedures

Reaction SMILES (text-based reaction representation, precursors>>products)

```
CC(=O)[O-].CC(=O)[O-].CC1=CCC(C)(C)c2cc(OS(=O)(=O)C(F)(F)F)c(C)cc21.CCOC(=O)c1ccc(N)cc1.Cc1ccccc1.O=C([O-])[O-].[Cs+].[Cs+].
[Pd+2].c1ccc(P(c2ccccc2)c2ccc3ccccc3c2-c2c(P(c3ccccc3)c3ccccc3)ccc3ccccc23)cc1>>CCOC(=O)c1ccc(Nc2cc3c(cc2C)C(C)=CCC3(C)C)cc1
```

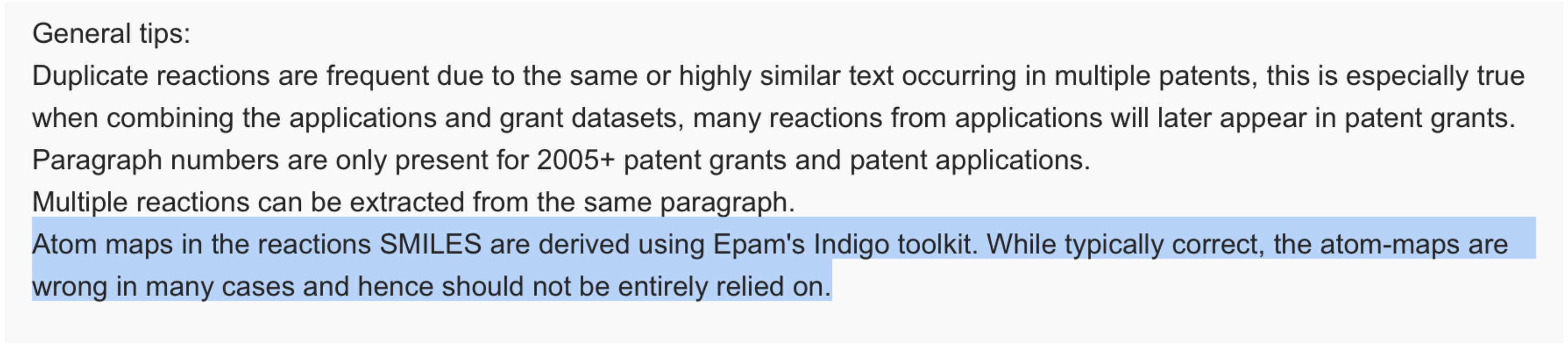
$\downarrow$ **Atom-mapping (e.g. RXNMapper)**

Atom-mapped reaction (required for reaction template, centre and bond change extraction)



```
CC(=O)[O-].CC(=O)[O-].Cc1ccccc1.O=C([O-])[O-].O=S(=O)(O[c:11]1[cH:12][c:13]2[c:14]([cH:15][c:16]1[CH3:17])[C:18]([CH3:19])=[CH:20]
[CH2:21][C:22]2([CH3:23])[CH3:24])C(F)(F)F.[CH3:1][CH2:2][O:3][C:4](=[O:5])[c:6]1[cH:7][cH:8][c:9]([NH2:10])[cH:25][cH:26]1.[Cs+].[Cs+].
[Pd+2].c1ccc(P(c2ccccc2)c2ccc3ccccc3c2-c2c(P(c3ccccc3)c3ccccc3)ccc3ccccc23)cc1>>[CH3:1][CH2:2][O:3][C:4](=[O:5])[c:6]1[cH:7][cH:8]
[c:9]([NH:10][c:11]2[cH:12][c:13]3[c:14]([cH:15][c:16]2[CH3:17])[C:18]([CH3:19])=[CH:20][CH2:21][C:22]3([CH3:23])[CH3:24])[cH:25][cH:26]1
```

Challenges with atom-mapping



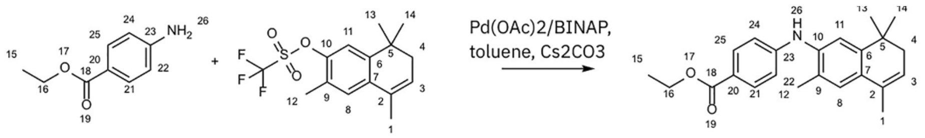
General tips:

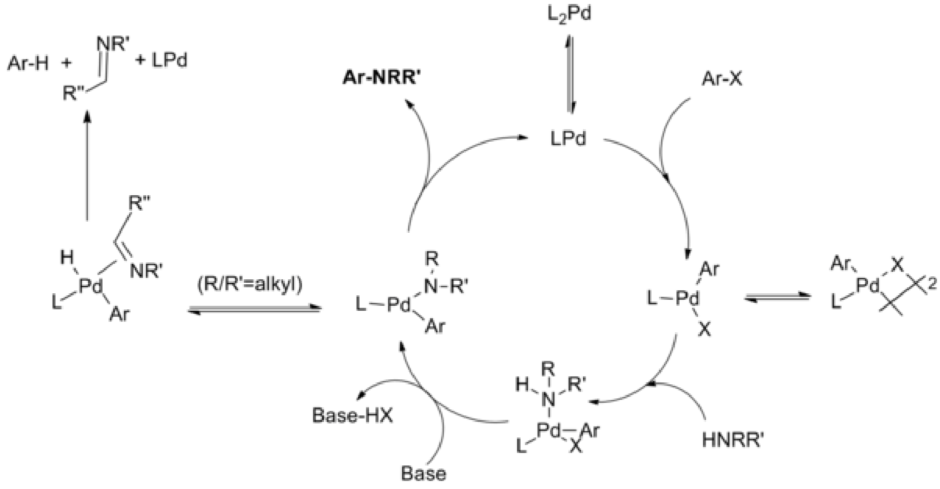
Duplicate reactions are frequent due to the same or highly similar text occurring in multiple patents, this is especially true when combining the applications and grant datasets, many reactions from applications will later appear in patent grants.

Paragraph numbers are only present for 2005+ patent grants and patent applications.

Multiple reactions can be extracted from the same paragraph.

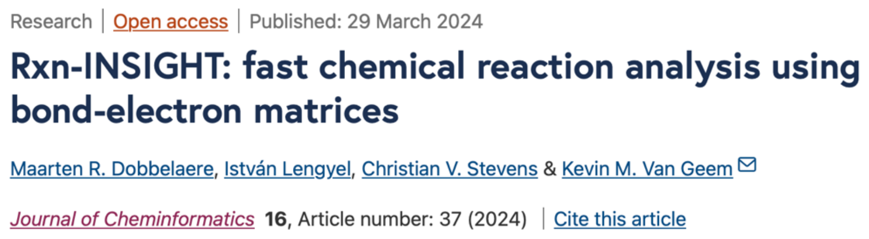
Atom maps in the reactions SMILES are derived using Epam's Indigo toolkit. While typically correct, the atom-maps are wrong in many cases and hence should not be entirely relied on.





Not taken into account with atom-mapping!

Reaction classes



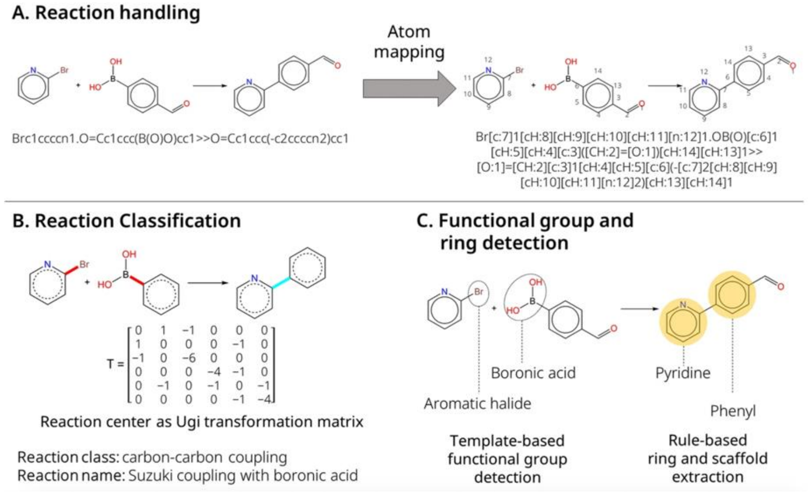
Research | Open access | Published: 29 March 2024

Rxn-INSIGHT: fast chemical reaction analysis using bond-electron matrices

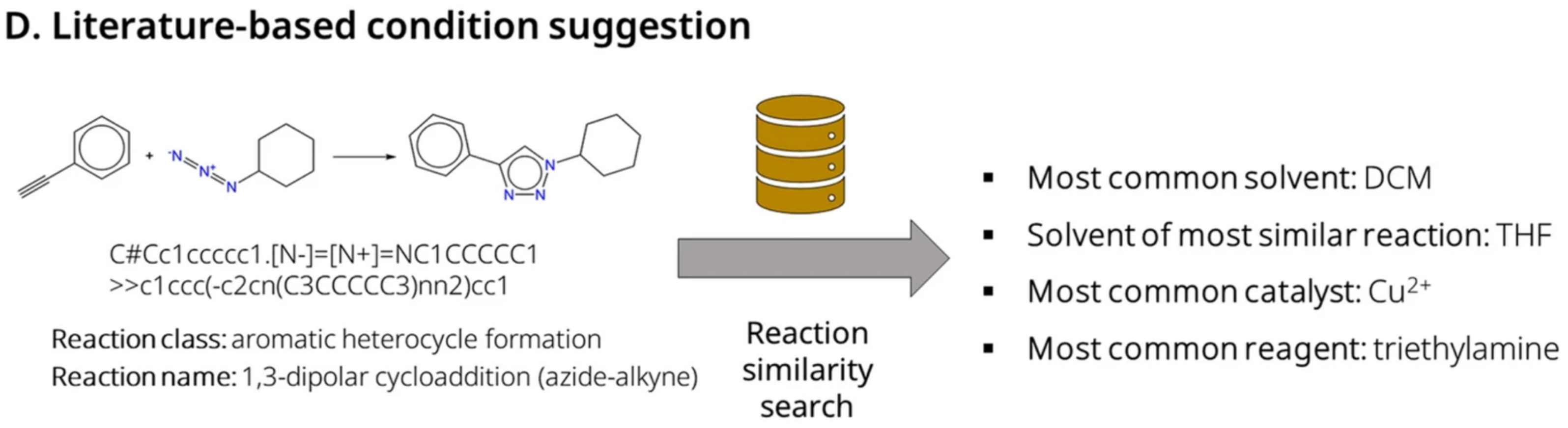
Maarten R. Dobbelaere, István Lengyel, Christian V. Stevens & Kevin M. Van Geem

Journal of Cheminformatics **16**, Article number: 37 (2024) | Cite this article

- Name reaction classes are typically assigned by closed source tools (e.g. NameRXN from Nextmove Software)
- New tool https://github.com/mrodobbe/Rxn-INSIGHT


A. Reaction handling
Atom mapping
Brc1ccccn1.O=Cc1ccc(B(O)O)cc1>>O=Cc1ccc(-c2ccccn2)cc1
Br[c:7]1[cH:8][cH:9][cH:10][cH:11][n:12]1.OB(O)[c:6]1[cH:5][cH:4][c:3]([CH:2]=[O:1])[cH:14][cH:13]1>>[O:1]=[CH:2][c:3]1[cH:4][cH:5][c:6](-[c:7]2[cH:8][cH:9][cH:10][cH:11][n:12]2)[cH:13][cH:14]1
B. Reaction Classification
$T = \begin{bmatrix} 0 & 1 & -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 & 0 \\ -1 & 0 & -6 & 0 & 0 & 0 \\ 0 & 0 & 0 & -4 & -1 & 0 \\ 0 & -1 & 0 & -1 & 0 & -1 \\ 0 & 0 & 0 & 0 & -1 & -4 \end{bmatrix}$
Reaction center as Ugi transformation matrix
Reaction class: carbon-carbon coupling
Reaction name: Suzuki coupling with boronic acid
C. Functional group and ring detection
Boronic acid
Aromatic halide
Pyridine
Phenyl
Template-based functional group detection
Rule-based ring and scaffold extraction

Reaction fingerprints

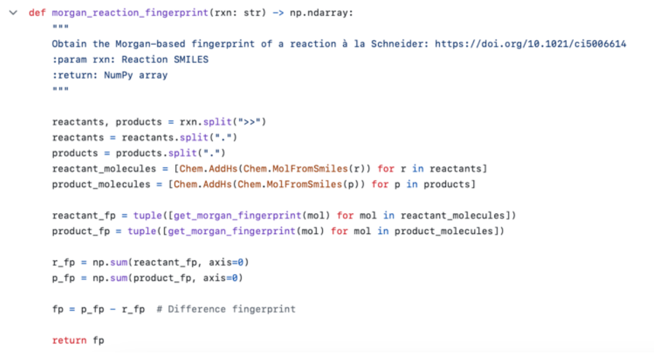

D. Literature-based condition suggestion

C#Cc1ccccc1.[N-]=[N+]=NC1CCCCC1
>>c1ccc(-c2cn(C3CCCCC3)nn2)cc1

Reaction class: aromatic heterocycle formation
Reaction name: 1,3-dipolar cycloaddition (azide-alkyne)

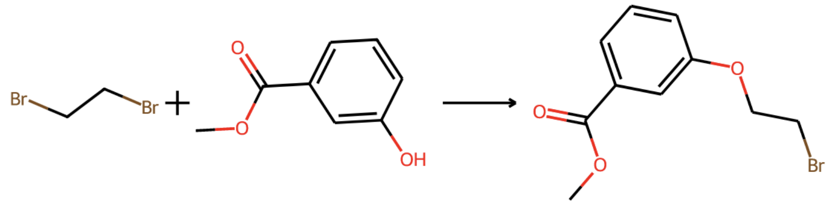
- Most common solvent: DCM
- Solvent of most similar reaction: THF
- Most common catalyst: Cu^{2+}
- Most common reagent: triethylamine

Either 166-bit MACCS keys [48] or 1024-bit extended-connectivity fingerprints with radius 2 (ECFP4; also: Morgan fingerprints) [49] can be used for representing the molecule. The reaction fingerprint is constructed by either adding or concatenating the molecular fingerprints of reactants and products [50].


```python
def morgan_reaction_fingerprint(rxn: str) -> np.ndarray:
    """
    Obtain the Morgan-based fingerprint of a reaction à la Schneider: https://doi.org/10.1021/ci5006614
    :param rxn: Reaction SMILES
    :return: NumPy array
    """

    reactants, products = rxn.split(">>")
    reactants = reactants.split(".")
    products = products.split(".")
    reactant_molecules = [Chem.AddHs(Chem.MolFromSmiles(r)) for r in reactants]
    product_molecules = [Chem.AddHs(Chem.MolFromSmiles(p)) for p in products]

    reactant_fp = tuple([get_morgan_fingerprint(mol) for mol in reactant_molecules])
    product_fp = tuple([get_morgan_fingerprint(mol) for mol in product_molecules])

    r_fp = np.sum(reactant_fp, axis=0)
    p_fp = np.sum(product_fp, axis=0)

    fp = p_fp - r_fp  # Difference fingerprint

    return fp
```

Does not handle reagents…

- https://github.com/mrodobbe/Rxn-INSIGHT/blob/master/rxn_insight/representation.py#L19

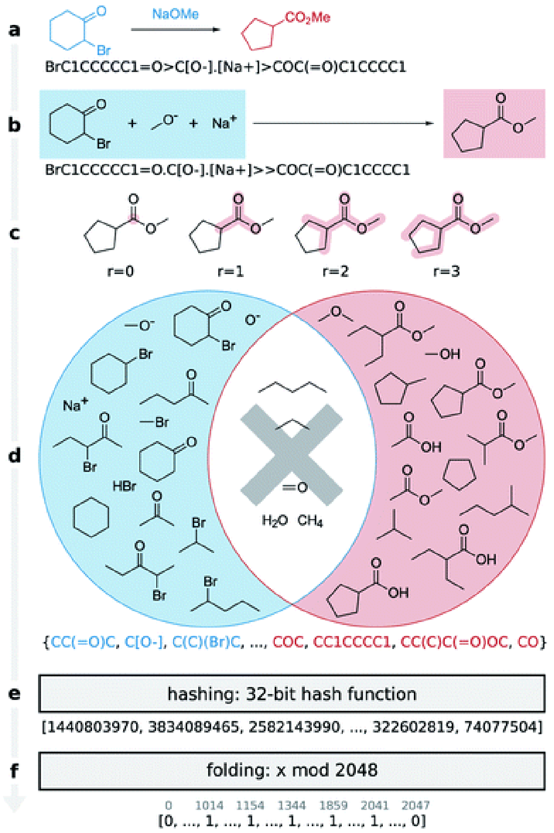


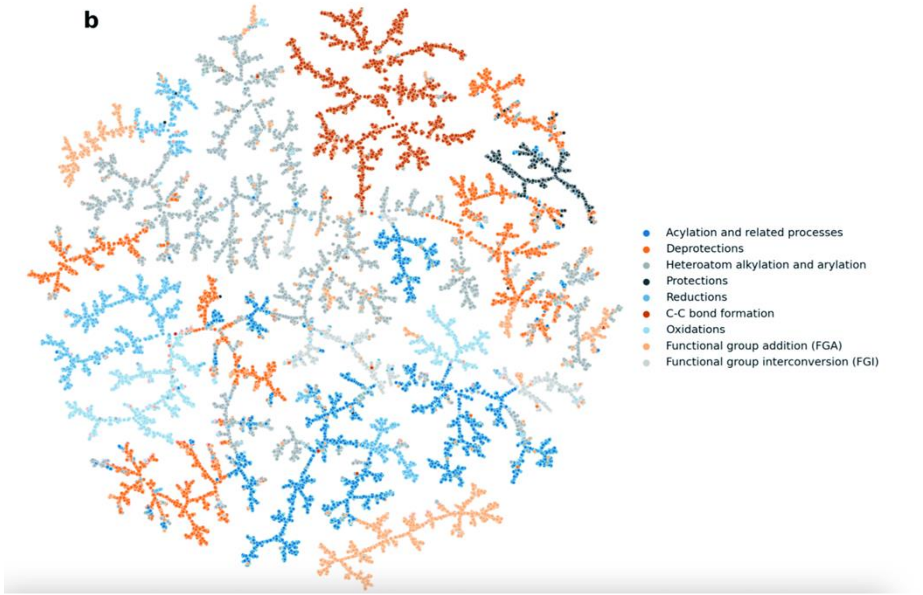
```
rxn = Reaction(r, solvent=solvent, reagent=reagent, catalyst=catalyst, ref=ref, rxn_mapper=rxn_mapper)
```

```
rxn.get_reaction_info()
```

```
{'REACTION': 'BrCCBr.COC(=O)c1cccc(O)c1>>COC(=O)c1cccc(OCCBr)c1',
 'MAPPED_REACTION': 'Br[CH2:11][CH2:12][Br:13].[CH3:1][O:2][C:3](=[O:4])[c:5]1[cH:6][cH:7][cH:8][c:9]([OH:10])[cH:14]1>>[CH3:1][O:2]
[C:3](=[O:4])[c:5]1[cH:6][cH:7][cH:8][c:9]([O:10][CH2:11][CH2:12][Br:13])[cH:14]1',
 'N_REACTANTS': 2,
 'N_PRODUCTS': 1,
 'FG_REACTANTS': ('Primary halide', 'Aromatic alcohol'),
 'FG_PRODUCTS': ('Ether',),
 'PARTICIPATING_RINGS_REACTANTS': (),
 'PARTICIPATING_RINGS_PRODUCTS': (),
 'ALL_RINGS_PRODUCTS': ('c1ccccc1',),
 'BY-PRODUCTS': ('HBr',),
 'CLASS': 'Heteroatom Alkylation and Arylation',
 'TAG': '7e1040116e3a4cf0e8ec5cc7865388d9cd0efabce6d320a1c25a36e411018d8b',
 'SOLVENT': ('CC(=O)C',),
 'REAGENT': ('',),
 'CATALYST': (),
 'REF': 'US08575180B2',
 'NAME': 'Williamson Ether Synthesis',
 'SCAFFOLD': 'c1ccccc1'}
```

Differential reaction fingerprint - DRFP





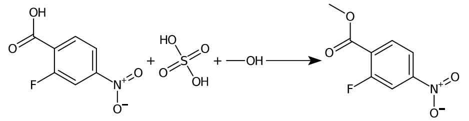
Heuristic-based bit fingerprint

Reaction classification and yield prediction using the differential reaction fingerprint DRFP

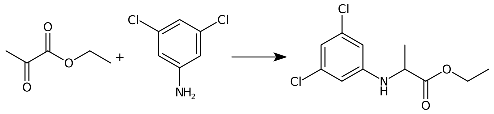
Daniel Probst *[a], Philippe Schwaller [b] and Jean-Louis Reymond *[a]

RXNFP

Reaction SMILES

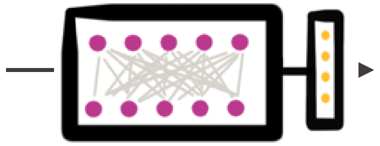


CO.O=C(O)c1ccc([N+](=O)[O-])cc1F.O=S(=O)(O)O>>COC(=O)c1ccc([N+](=O)[O-])cc1F



CCOC(=O)C(C)=O.Nc1cc(Cl)cc(Cl)c1>>CCOC(=O)C(C)Nc1cc(Cl)cc(Cl)c1

Classification model



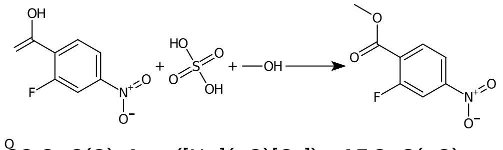
Reaction class

Fischer-Speier Esterification 2.6.3

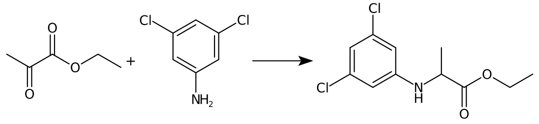
Ketone reductive amination 1.2.5

>98% accurate

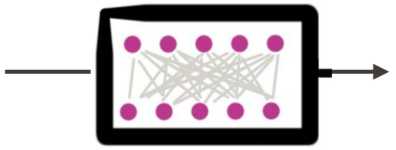
Reaction encoder



CO.O=C(O)c1ccc([N+](=O)[O-])cc1F.O=S(=O)(O)O>>COC(=O)c1ccc([N+](=O)[O-])cc1F



CCOC(=O)C(C)=O.Nc1cc(Cl)cc(Cl)c1>>CCOC(=O)C(C)Nc1cc(Cl)cc(Cl)c1



Fingerprints

= Encoded reaction properties

[0.14, 0.25, ..., 0.9]

[0.22, 0.83, ..., -0.12]

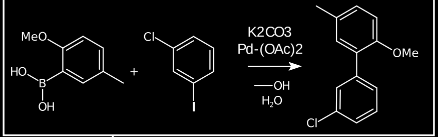
Mapping the space of chemical reactions using attention-based neural networks

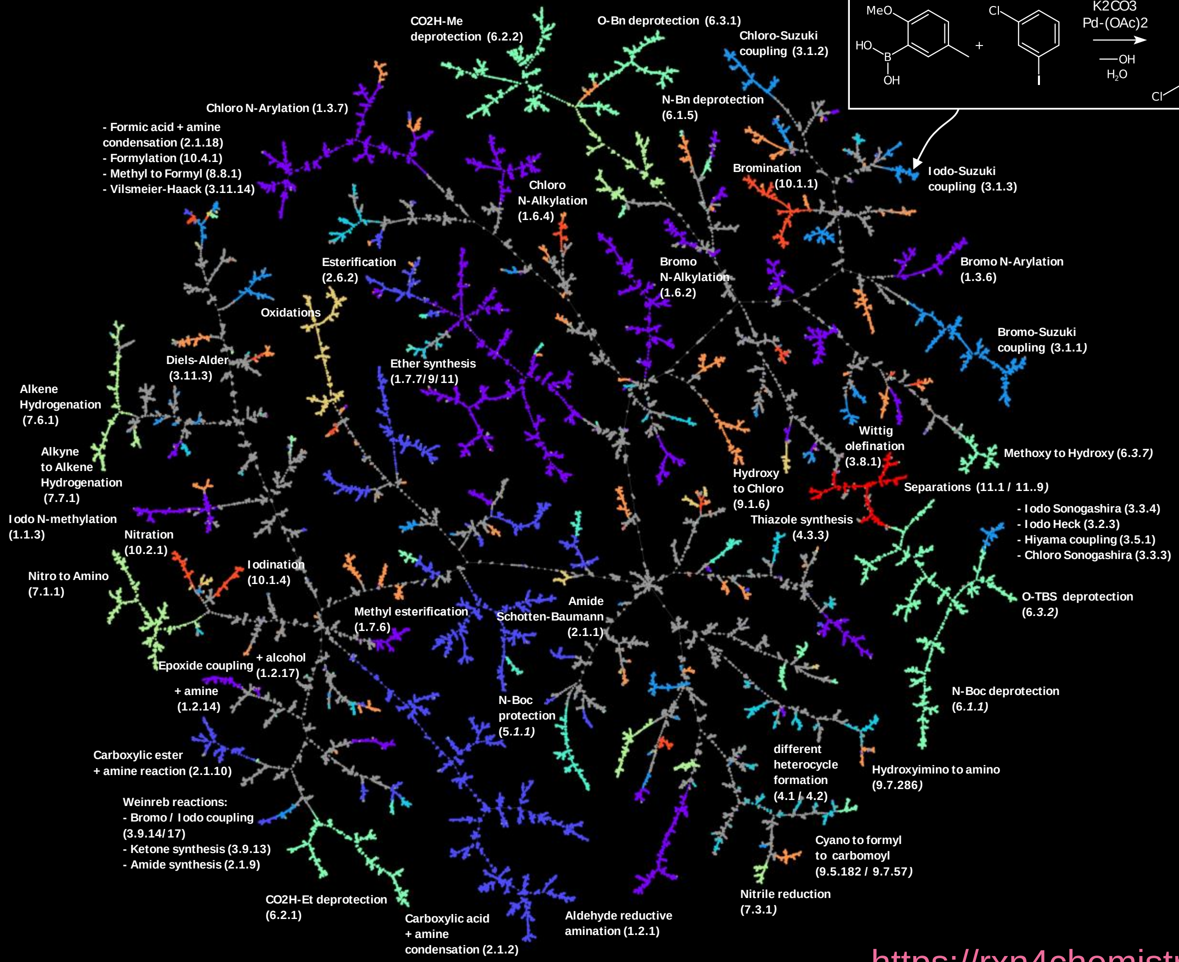
Philippe Schwaller[1,2], Daniel Probst[2], Alain C. Vaucher[1], Vishnu H. Nair[1], David Kreutter[2], Teodoro Laino[1] and Jean-Louis Reymond[2]

nature machine intelligence

Chemical Reaction Atlas

Iodo-Suzuki coupling





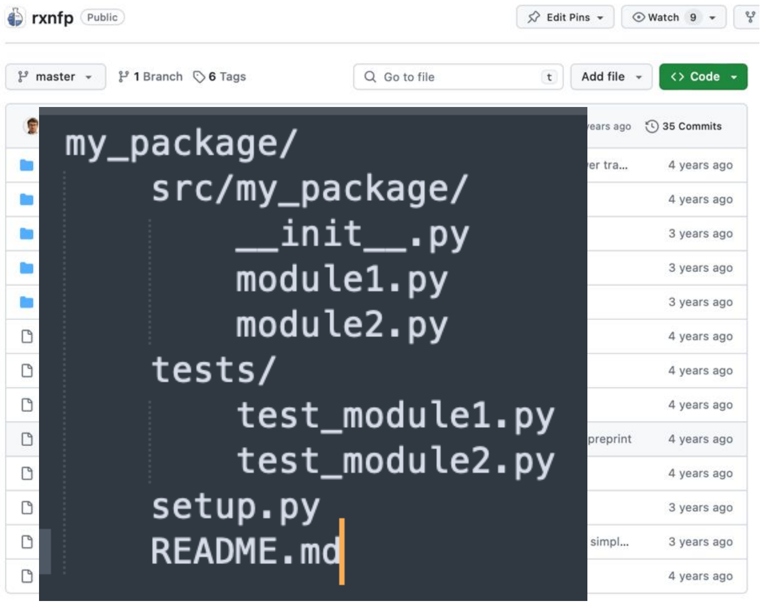
Powered by TMAP
Probst & Reymond

- ⬜ Unrecognised
- 🟪 Heteroatom alkylation and arylation
- 🟦 Acylation and related processes
- 🟦 C-C bond formation
- 🟦 Heterocycle formation
- 🟩 Protections
- 🟩 Deprotections
- 🟩 Reductions
- 🟨 Oxidations
- 🟧 Functional group interconvertions
- 🟧 Functional group additions
- 🟥 Resolutions

https://rxn4chemistry.github.io/rxnfp//tmaps/tmap_ft_10k.htm

Python packages
-- sharing reusable functionality

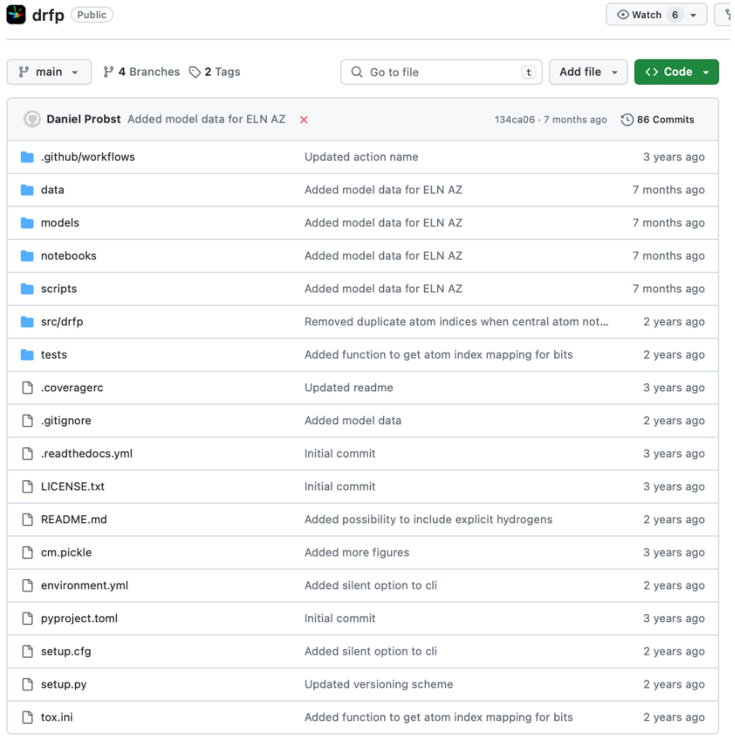
If we look at those packages



```
my_package/
    src/my_package/
        __init__.py
        module1.py
        module2.py
    tests/
        test_module1.py
        test_module2.py
    setup.py
    README.md
```

ears ago	
er tra...	4 years ago
	4 years ago
	3 years ago
	3 years ago
	3 years ago
	4 years ago
	4 years ago
	4 years ago
preprint	4 years ago
	4 years ago
	3 years ago
simpl...	3 years ago
	4 years ago

https://github.com/rxn4chemistry/rxnfp

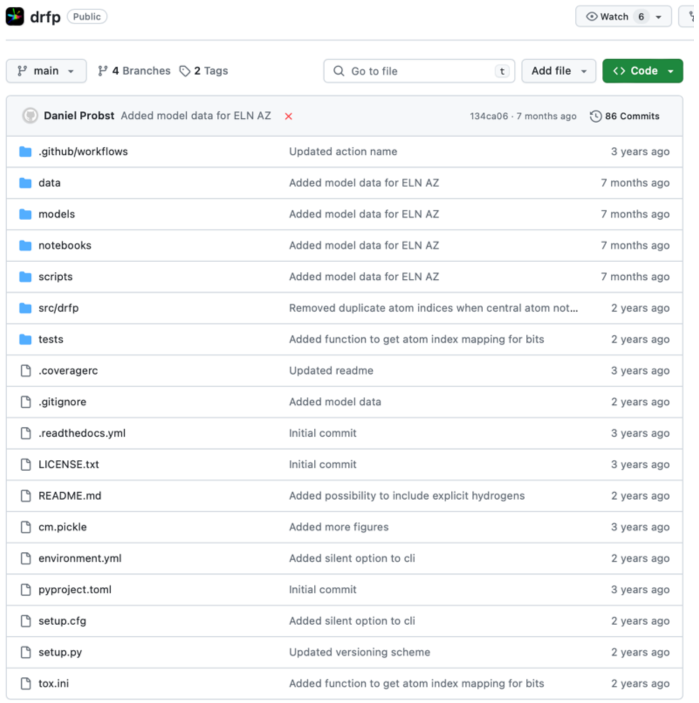


Daniel Probst Added model data for ELN AZ × 134ca06 · 7 months ago 86 Commits

.github/workflows	Updated action name	3 years ago
data	Added model data for ELN AZ	7 months ago
models	Added model data for ELN AZ	7 months ago
notebooks	Added model data for ELN AZ	7 months ago
scripts	Added model data for ELN AZ	7 months ago
src/drfp	Removed duplicate atom indices when central atom not...	2 years ago
tests	Added function to get atom index mapping for bits	2 years ago
.coveragerc	Updated readme	3 years ago
.gitignore	Added model data	2 years ago
.readthedocs.yml	Initial commit	3 years ago
LICENSE.txt	Initial commit	3 years ago
README.md	Added possibility to include explicit hydrogens	2 years ago
cm.pickle	Added more figures	3 years ago
environment.yml	Added silent option to cli	2 years ago
pyproject.toml	Initial commit	3 years ago
setup.cfg	Added silent option to cli	2 years ago
setup.py	Updated versioning scheme	2 years ago
tox.ini	Added function to get atom index mapping for bits	2 years ago

https://github.com/reymond-group/drfp

Intimidating? Yes. Let's look at it in more detail.

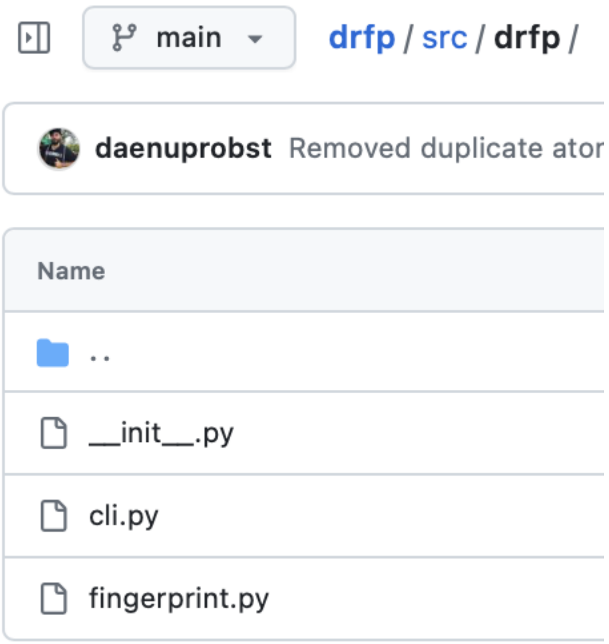


Daniel Probst Added model data for ELN AZ 134ca06 · 7 months ago 86 Commits

.github/workflows	Updated action name	3 years ago
data	Added model data for ELN AZ	7 months ago
models	Added model data for ELN AZ	7 months ago
notebooks	Added model data for ELN AZ	7 months ago
scripts	Added model data for ELN AZ	7 months ago
src/drfp	Removed duplicate atom indices when central atom not...	2 years ago
tests	Added function to get atom index mapping for bits	2 years ago
.coveragerc	Updated readme	3 years ago
.gitignore	Added model data	2 years ago
.readthedocs.yml	Initial commit	3 years ago
LICENSE.txt	Initial commit	3 years ago
README.md	Added possibility to include explicit hydrogens	2 years ago
cm.pickle	Added more figures	3 years ago
environment.yml	Added silent option to cli	2 years ago
pyproject.toml	Initial commit	3 years ago
setup.cfg	Added silent option to cli	2 years ago
setup.py	Updated versioning scheme	2 years ago
tox.ini	Added function to get atom index mapping for bits	2 years ago

- **.github/workflows (optional):** GitHub Actions workflows, which are automated scripts that you can set up
- **data (optional):** Data needed for your package's operations
- **models (optional):** Files related to data models (e.g. machine learning models)
- **notebooks (optional):** Jupyter notebooks that make use of your package.
- **scripts (optional):** Executable scripts that perform tasks using your package.
- **src/drfp (the main folder):** Short for "source", this directory will contain the actual Python source code of the package.
- **tests (recommended):** Contains test code. This directory will have unit tests that help ensure your code is running correctly.

The source folder → src/drfp


main
drfp / src / drfp /
daenuprobst Removed duplicate ator
Name
..
__init__.py
cli.py
fingerprint.py

The presence of an **__init__.py** file in a directory indicates to Python that the **directory should be treated as a package**. This means that the directory's name can be used in import statements.
Typically, empty but you can use it to simplify imports.

cli stands for **c**ommand **l**ine **i**nterface, it contains the code to make the package work from the command line (optional).

```
drfp my_rxn_smiles.txt my_rxn_fps.pkl -d 512
```

Input file

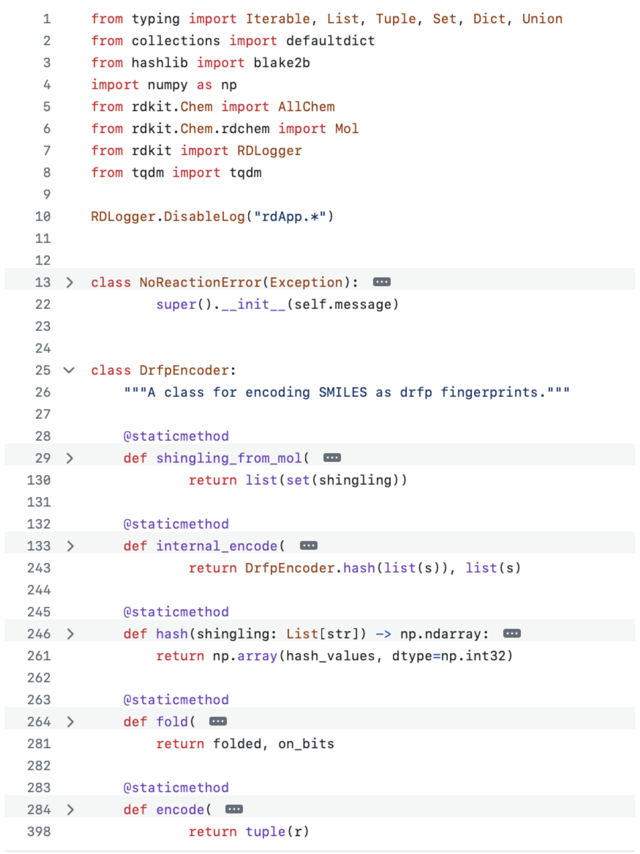
Output file

Extra argument for fp dimensions

fingerprints.py is where all the magic happens
→ the package functionality is written.

fingerprint is the name of this module in the drfp package.

fingerprint.py file



```python
from typing import Iterable, List, Tuple, Set, Dict, Union
from collections import defaultdict
from hashlib import blake2b
import numpy as np
from rdkit.Chem import AllChem
from rdkit.Chem.rdchem import Mol
from rdkit import RDLogger
from tqdm import tqdm

RDLogger.DisableLog("rdApp.*")


class NoReactionError(Exception): ⋯
        super().__init__(self.message)


class DrfpEncoder:
    """A class for encoding SMILES as drfp fingerprints."""

    @staticmethod
    def shingling_from_mol( ⋯
            return list(set(shingling))

    @staticmethod
    def internal_encode( ⋯
            return DrfpEncoder.hash(list(s)), list(s)

    @staticmethod
    def hash(shingling: List[str]) -> np.ndarray: ⋯
        return np.array(hash_values, dtype=np.int32)

    @staticmethod
    def fold( ⋯
        return folded, on_bits

    @staticmethod
    def encode( ⋯
            return tuple(r)
```

Imports from other packages

Silence Rdkit warnings (optional)

Defining an exception (optional)

The DrfpEncoder class, which contains all the functionality to generate fingerprints from reaction SMILES.

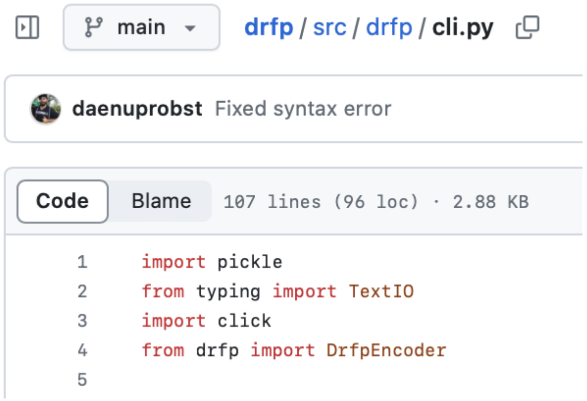
@static_method
→ Function that does not require data from class (no self)

The main function is “encode”

```python
def encode(
    X: Union[Iterable, str],
    n_folded_length: int = 2048,
    min_radius: int = 0,
    radius: int = 3,
    rings: bool = True,
    mapping: bool = False,
    atom_index_mapping: bool = False,
    root_central_atom: bool = True,
    include_hydrogens: bool = False,
    show_progress_bar: bool = False,
) -> Union[
    List[np.ndarray],
    Tuple[List[np.ndarray], Dict[int, Set[str]]],
    Tuple[List[np.ndarray], Dict[int, Set[str]]],
    List[Dict[str, List[Dict[str, List[Set[int]]]]]],
]:
    """Encodes a list of reaction SMILES using the drfp fingerprint.

    Args:
        X: An iterable (e.g. List) of reaction SMILES or a single reaction SMILES to be encoded
        n_folded_length: The folded length of the fingerprint (the parameter for the modulo hashing)
        min_radius: The minimum radius of a substructure (0 includes single atoms)
        radius: The maximum radius of a substructure
        rings: Whether to include full rings as substructures
        mapping: Return a feature to substructure mapping in addition to the fingerprints
        atom_index_mapping: Return the atom indices of mapped substructures for each reaction
        root_central_atom: Whether to root the central atom of substructures when generating SMILES
        show_progress_bar: Whether to show a progress bar when encoding reactions

    Returns:
        A list of drfp fingerprints or, if mapping is enabled, a tuple containing a list of drfp fingerprints and a mapping dict.
    """
```

Command line interface – cli.py

```
drfp my_rxn_smiles.txt my_rxn_fps.pkl -d 512
```


main
drfp / src / drfp / cli.py
daenuprobst Fixed syntax error
Code
Blame
107 lines (96 loc) · 2.88 KB

```python
import pickle
from typing import TextIO
import click
from drfp import DrfpEncoder

```

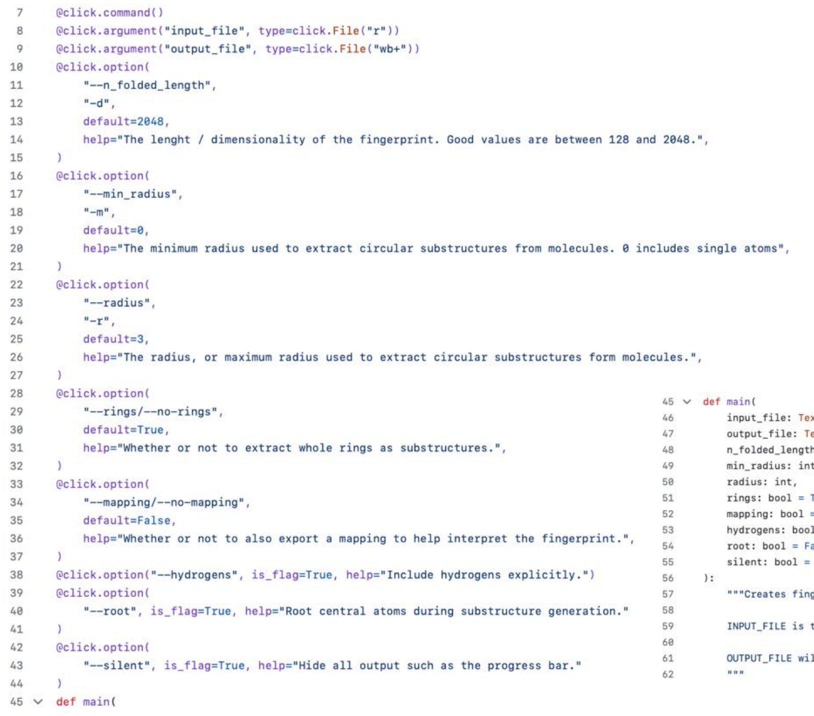
Click is a Python package for creating beautiful command line interfaces in a composable way with as little code as necessary. It's the "Command Line Interface Creation Kit". It's highly configurable but comes with sensible defaults out of the box.

https://click.palletsprojects.com/en/8.1.x/

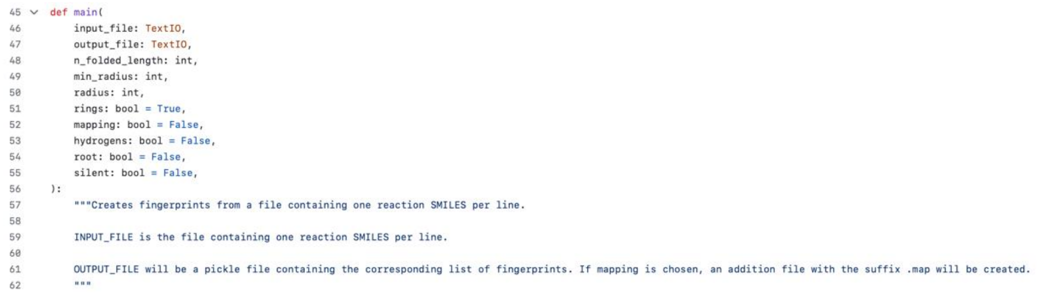
https://github.com/reymond-group/drfp/blob/main/src/drfp/cli.py

Command line interface – cli.py

```
drfp my_rxn_smiles.txt my_rxn_fps.pkl -d 512
```



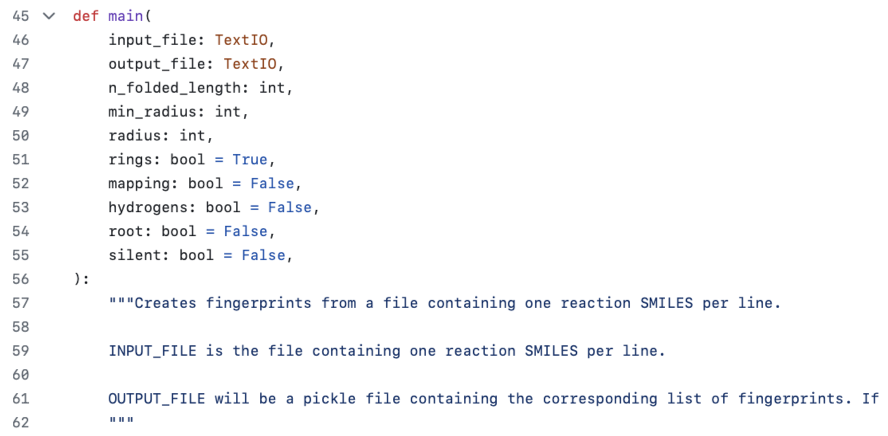
```python
@click.command()
@click.argument("input_file", type=click.File("r"))
@click.argument("output_file", type=click.File("wb+"))
@click.option(
    "--n_folded_length",
    "-d",
    default=2048,
    help="The lenght / dimensionality of the fingerprint. Good values are between 128 and 2048.",
)
@click.option(
    "--min_radius",
    "-m",
    default=0,
    help="The minimum radius used to extract circular substructures from molecules. 0 includes single atoms",
)
@click.option(
    "--radius",
    "-r",
    default=3,
    help="The radius, or maximum radius used to extract circular substructures form molecules.",
)
@click.option(
    "--rings/--no-rings",
    default=True,
    help="Whether or not to extract whole rings as substructures.",
)
@click.option(
    "--mapping/--no-mapping",
    default=False,
    help="Whether or not to also export a mapping to help interpret the fingerprint.",
)
@click.option("--hydrogens", is_flag=True, help="Include hydrogens explicitly.")
@click.option(
    "--root", is_flag=True, help="Root central atoms during substructure generation."
)
@click.option(
    "--silent", is_flag=True, help="Hide all output such as the progress bar."
)
def main(
```



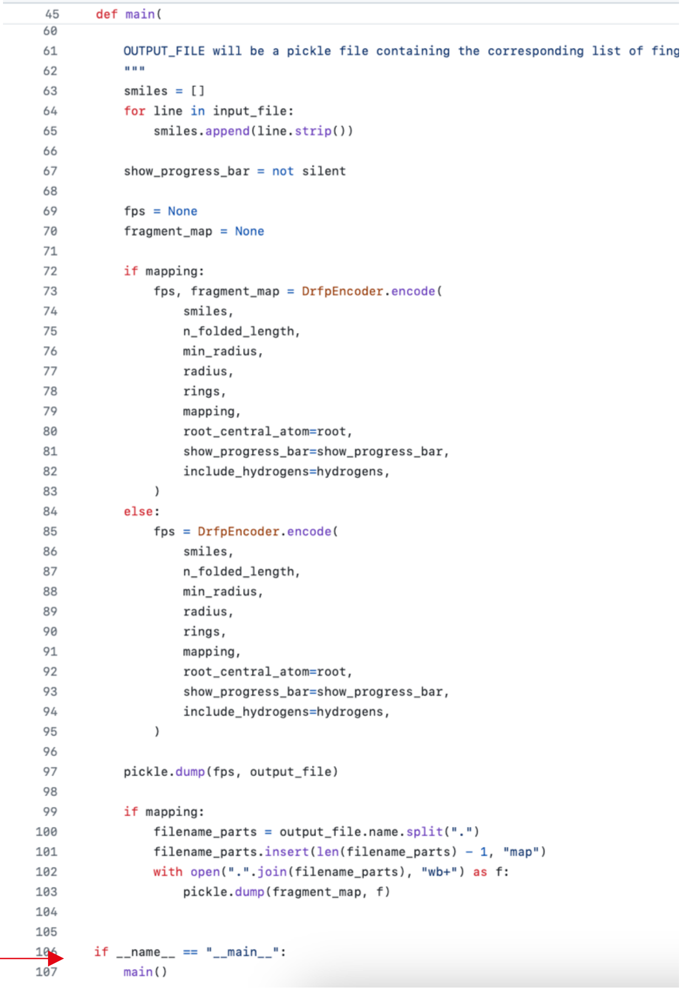
```python
def main(
    input_file: TextIO,
    output_file: TextIO,
    n_folded_length: int,
    min_radius: int,
    radius: int,
    rings: bool = True,
    mapping: bool = False,
    hydrogens: bool = False,
    root: bool = False,
    silent: bool = False,
):
    """Creates fingerprints from a file containing one reaction SMILES per line.

    INPUT_FILE is the file containing one reaction SMILES per line.

    OUTPUT_FILE will be a pickle file containing the corresponding list of fingerprints. If mapping is chosen, an addition file with the suffix .map will be created.
    """
```

https://github.com/reymond-group/drfp/blob/main/src/drfp/cli.py

```
drfp my_rxn_smiles.txt my_rxn_fps.pkl -d 512
```



```
def main(
    input_file: TextIO,
    output_file: TextIO,
    n_folded_length: int,
    min_radius: int,
    radius: int,
    rings: bool = True,
    mapping: bool = False,
    hydrogens: bool = False,
    root: bool = False,
    silent: bool = False,
):
    """Creates fingerprints from a file containing one reaction SMILES per line.

    INPUT_FILE is the file containing one reaction SMILES per line.

    OUTPUT_FILE will be a pickle file containing the corresponding list of fingerprints. If
    """
```



```
def main(

    OUTPUT_FILE will be a pickle file containing the corresponding list of fing
    """
    smiles = []
    for line in input_file:
        smiles.append(line.strip())

    show_progress_bar = not silent

    fps = None
    fragment_map = None

    if mapping:
        fps, fragment_map = DrfpEncoder.encode(
            smiles,
            n_folded_length,
            min_radius,
            radius,
            rings,
            mapping,
            root_central_atom=root,
            show_progress_bar=show_progress_bar,
            include_hydrogens=hydrogens,
        )
    else:
        fps = DrfpEncoder.encode(
            smiles,
            n_folded_length,
            min_radius,
            radius,
            rings,
            mapping,
            root_central_atom=root,
            show_progress_bar=show_progress_bar,
            include_hydrogens=hydrogens,
        )

    pickle.dump(fps, output_file)

    if mapping:
        filename_parts = output_file.name.split(".")
        filename_parts.insert(len(filename_parts) - 1, "map")
        with open(".".join(filename_parts), "wb+") as f:
            pickle.dump(fragment_map, f)


if __name__ == "__main__":
    main()
```

- The **if __name__ == "__main__":** statement in Python is used to determine whether a Python script is being run directly or being imported into another script as a module. Let’s break it down.
- **__name__**: This is a special built-in variable in Python. It represents the name of the current module. However, if a **module is being run directly** (i.e., not imported from another script), **__name__** is set to the string **"__main__"** by the Python interpreter.
- **if __name__ == "__main__":** This line checks if the script is being run directly. If this condition is **True**, it **means the script is not being imported** and is the main program being executed.
- **main()**: This is a call to a function named main(), which you would define elsewhere in your script. This function typically contains the code that you want to run when the script is executed directly.

Code is run through **the terminal using “python example_file.py”** → if statement is **executed** → CLI

Code is imported in another script/notebook using ”**from example_file import exciting_function”**
→ if statement **not executed**.

EPFL

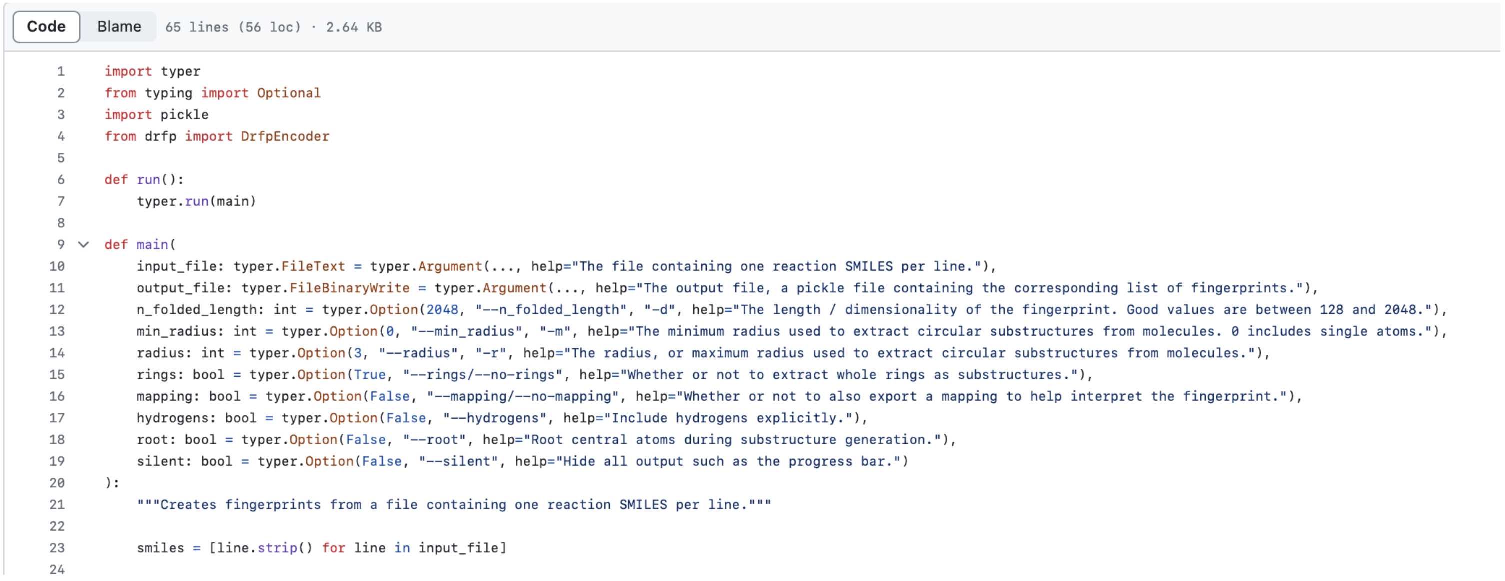
There is never just one solution. Another maybe simpler CLI.

Typer

- https://github.com/tiangolo/typer

Typer, build great CLIs. Easy to code. Based on Python type hints.

Test passing | Publish passing | coverage 100% | pypi package v0.12.0



Code | Blame | 65 lines (56 loc) · 2.64 KB

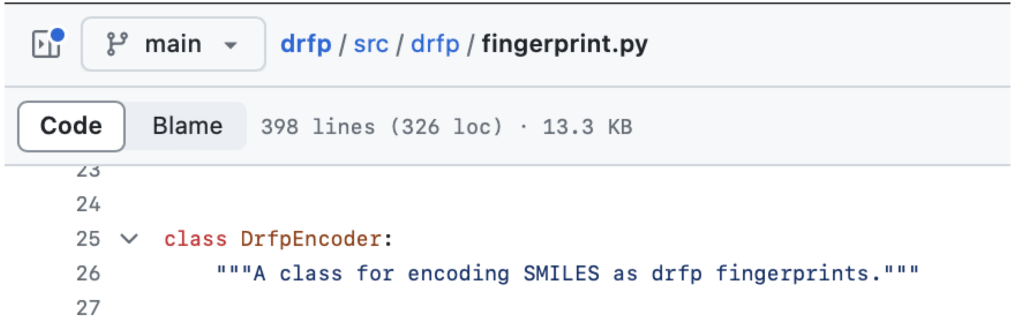
```python
import typer
from typing import Optional
import pickle
from drfp import DrfpEncoder

def run():
    typer.run(main)

def main(
    input_file: typer.FileText = typer.Argument(..., help="The file containing one reaction SMILES per line."),
    output_file: typer.FileBinaryWrite = typer.Argument(..., help="The output file, a pickle file containing the corresponding list of fingerprints."),
    n_folded_length: int = typer.Option(2048, "--n_folded_length", "-d", help="The length / dimensionality of the fingerprint. Good values are between 128 and 2048."),
    min_radius: int = typer.Option(0, "--min_radius", "-m", help="The minimum radius used to extract circular substructures from molecules. 0 includes single atoms."),
    radius: int = typer.Option(3, "--radius", "-r", help="The radius, or maximum radius used to extract circular substructures from molecules."),
    rings: bool = typer.Option(True, "--rings/--no-rings", help="Whether or not to extract whole rings as substructures."),
    mapping: bool = typer.Option(False, "--mapping/--no-mapping", help="Whether or not to also export a mapping to help interpret the fingerprint."),
    hydrogens: bool = typer.Option(False, "--hydrogens", help="Include hydrogens explicitly."),
    root: bool = typer.Option(False, "--root", help="Root central atoms during substructure generation."),
    silent: bool = typer.Option(False, "--silent", help="Hide all output such as the progress bar.")
):
    """Creates fingerprints from a file containing one reaction SMILES per line."""

    smiles = [line.strip() for line in input_file]

```

- https://github.com/pschwllr/drfp/blob/main/src/drfp/cli.py
  https://github.com/reymond-group/drfp/compare/main...pschwllr:drfp:main

Simplify imports in the __init__.py file (advanced)

In the example,
in the README.

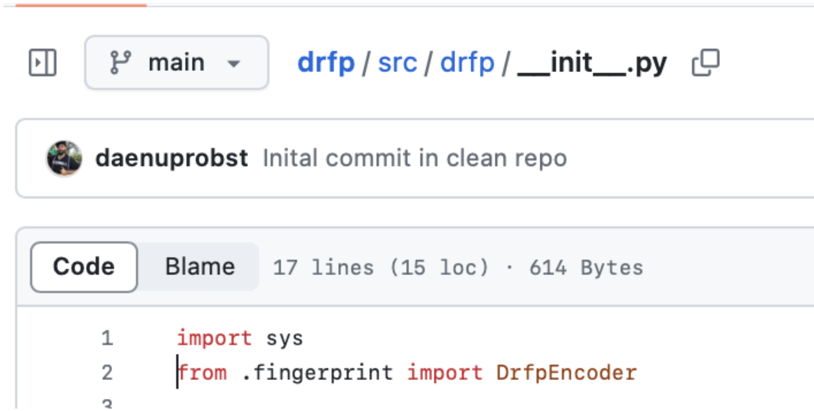
```python
from drfp import DrfpEncoder

rxn_smiles = [
    "CO.O[C@@H]1CCNC1.[C-]#[N+]CC(=O)OC>>[C-]#[N+]CC(=O)N1CC[C@@H](O)C1",
    "CCOC(=O)C(CC)c1cccnc1.Cl.O>>CCC(C(=O)O)c1cccnc1",
]

fps = DrfpEncoder.encode(rxn_smiles)
```



main drfp / src / drfp / fingerprint.py

Code Blame 398 lines (326 loc) · 13.3 KB

```python

class DrfpEncoder:
    """A class for encoding SMILES as drfp fingerprints."""

```



main drfp / src / drfp / __init__.py

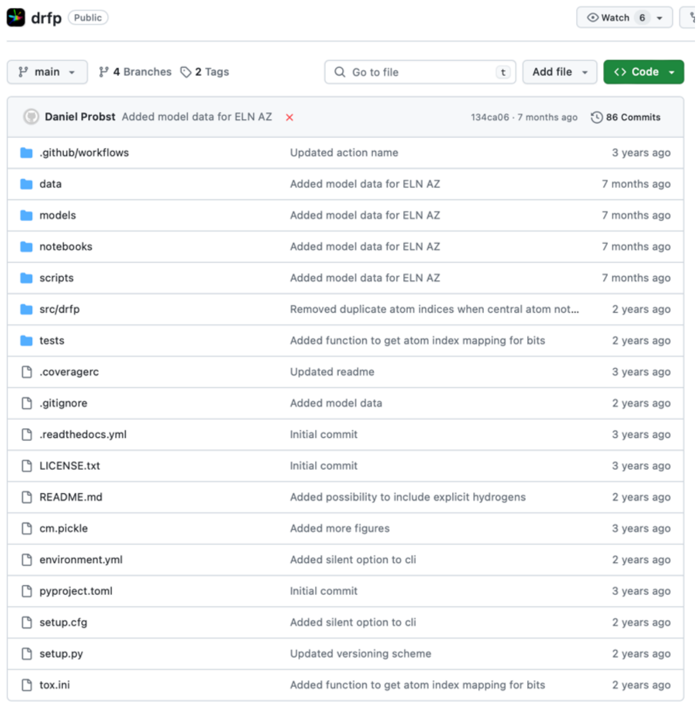
daenuprobst Inital commit in clean repo

Code Blame 17 lines (15 loc) · 614 Bytes

```python
import sys
from .fingerprint import DrfpEncoder
```

Why not "**from drfp.fingerprint import DrfpEncoder**" ?

What are all the other files?



Daniel Probst Added model data for ELN AZ 134ca06 · 7 months ago 86 Commits

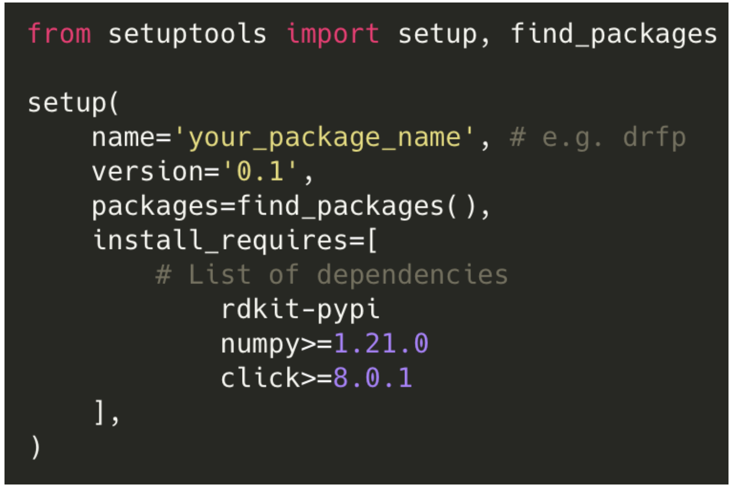
.github/workflows	Updated action name	3 years ago
data	Added model data for ELN AZ	7 months ago
models	Added model data for ELN AZ	7 months ago
notebooks	Added model data for ELN AZ	7 months ago
scripts	Added model data for ELN AZ	7 months ago
src/drfp	Removed duplicate atom indices when central atom not...	2 years ago
tests	Added function to get atom index mapping for bits	2 years ago
.coveragerc	Updated readme	3 years ago
.gitignore	Added model data	2 years ago
.readthedocs.yml	Initial commit	3 years ago
LICENSE.txt	Initial commit	3 years ago
README.md	Added possibility to include explicit hydrogens	2 years ago
cm.pickle	Added more figures	3 years ago
environment.yml	Added silent option to cli	2 years ago
pyproject.toml	Initial commit	3 years ago
setup.cfg	Added silent option to cli	2 years ago
setup.py	Updated versioning scheme	2 years ago
tox.ini	Added function to get atom index mapping for bits	2 years ago

- **.coveragerc (optional)**: A configuration file for coverage.py, a tool for measuring code coverage of Python programs.
- **.gitignore (super useful)**: A Git configuration file that tells Git which files or directories to ignore in a project (temporary files, build artifacts, etc.).
- **.readthedocs.yml (optional)**: Configuration file for Read the Docs, a documentation hosting platform that can automatically build and host your package's documentation.
- **LICENSE.txt (important)**: The license file specifies the terms under which your package is made available. It's important for defining how others can use your code.
- **README.md (important)**: A Markdown file that provides an introduction and overview of your package. What appears on the github repo.
- **environment.yml (optional)**: If this project is set up to work with conda, this YAML file will specify the package dependencies for the project so that a user can recreate the package's environment.
- **pyproject.toml (important)**: A configuration file for build system requirements for Python projects
- **setup.cfg (old school)**: A setup configuration file, which is used to specify metadata and configuration parameters for the package, like package name, version, and dependencies.
- **setup.py (old school)**: A Python script that serves as the build script for setuptools. It tells setuptools about your package (such as the name and version) as well as which code files to include.
- **tox.ini (optional)**: A configuration file for tox, a tool for testing your code on multiple Python environments.

How to tell pip how to build your package?

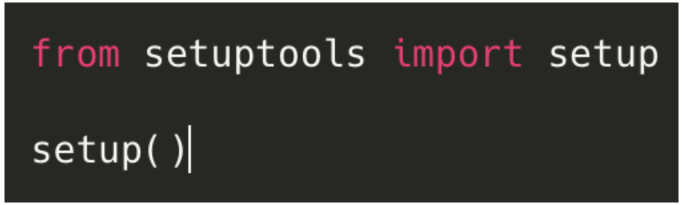
setup.py (old school) and setup.cfg (separate static metadata) – learn to understand

setup.py only



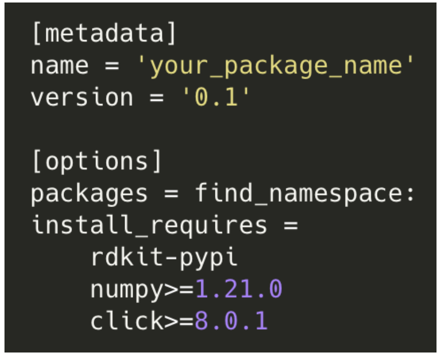
```
from setuptools import setup, find_packages

setup(
    name='your_package_name', # e.g. drfp
    version='0.1',
    packages=find_packages(),
    install_requires=[
        # List of dependencies
            rdkit-pypi
            numpy>=1.21.0
            click>=8.0.1
    ],
)
```

setup.py (automatically reads setup.cfg)



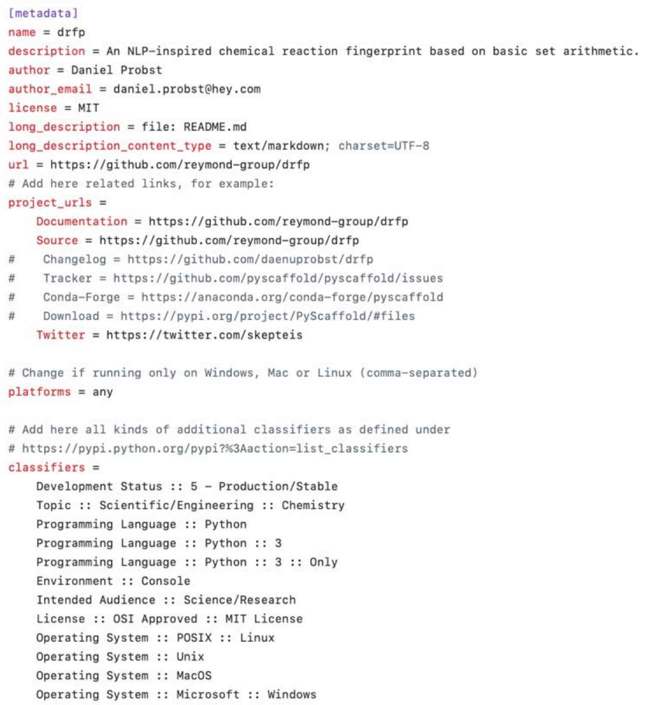
```
from setuptools import setup

setup()
```

and setup.cfg

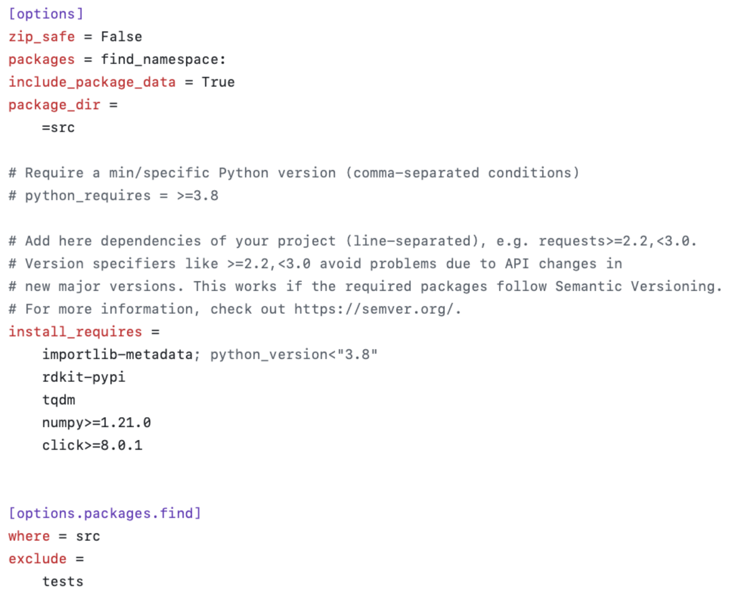


```
[metadata]
name = 'your_package_name'
version = '0.1'

[options]
packages = find_namespace:
install_requires =
    rdkit-pypi
    numpy>=1.21.0
    click>=8.0.1
```

There is a lot more information that can be put into those files (.cfg)



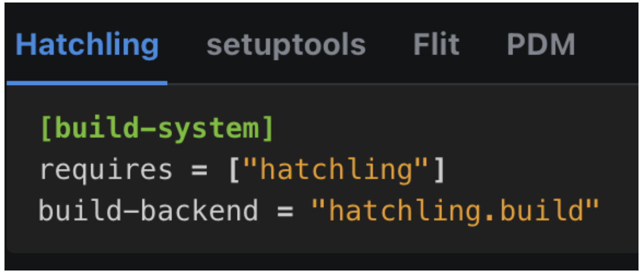
```
[metadata]
name = drfp
description = An NLP-inspired chemical reaction fingerprint based on basic set arithmetic.
author = Daniel Probst
author_email = daniel.probst@hey.com
license = MIT
long_description = file: README.md
long_description_content_type = text/markdown; charset=UTF-8
url = https://github.com/reymond-group/drfp
# Add here related links, for example:
project_urls =
    Documentation = https://github.com/reymond-group/drfp
    Source = https://github.com/reymond-group/drfp
#    Changelog = https://github.com/daenuprobst/drfp
#    Tracker = https://github.com/pyscaffold/pyscaffold/issues
#    Conda-Forge = https://anaconda.org/conda-forge/pyscaffold
#    Download = https://pypi.org/project/PyScaffold/#files
    Twitter = https://twitter.com/skepteis

# Change if running only on Windows, Mac or Linux (comma-separated)
platforms = any

# Add here all kinds of additional classifiers as defined under
# https://pypi.python.org/pypi?%3Aaction=list_classifiers
classifiers =
    Development Status :: 5 - Production/Stable
    Topic :: Scientific/Engineering :: Chemistry
    Programming Language :: Python
    Programming Language :: Python :: 3
    Programming Language :: Python :: 3 :: Only
    Environment :: Console
    Intended Audience :: Science/Research
    License :: OSI Approved :: MIT License
    Operating System :: POSIX :: Linux
    Operating System :: Unix
    Operating System :: MacOS
    Operating System :: Microsoft :: Windows
```



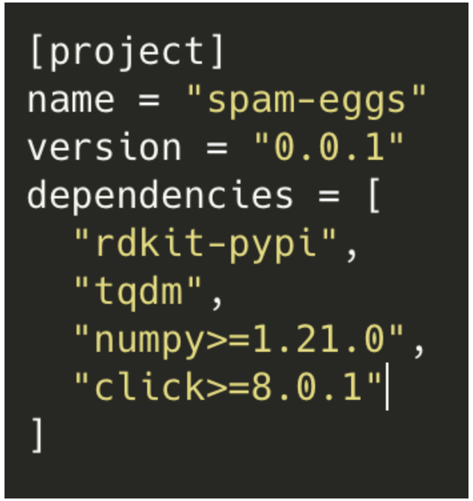
```
[options]
zip_safe = False
packages = find_namespace:
include_package_data = True
package_dir =
    =src

# Require a min/specific Python version (comma-separated conditions)
# python_requires = >=3.8

# Add here dependencies of your project (line-separated), e.g. requests>=2.2,<3.0.
# Version specifiers like >=2.2,<3.0 avoid problems due to API changes in
# new major versions. This works if the required packages follow Semantic Versioning.
# For more information, check out https://semver.org/.
install_requires =
    importlib-metadata; python_version<"3.8"
    rdkit-pypi
    tqdm
    numpy>=1.21.0
    click>=8.0.1


[options.packages.find]
where = src
exclude =
    tests
```

pyproject.toml (the new recommended way)

- 1 single file ”pyproject.toml
- https://packaging.python.org/en/latest/guides/writing-pyproject-toml/



```
[build-system]
requires = ["hatchling"]
build-backend = "hatchling.build"
```



```
[project]
name = "spam-eggs"
version = "0.0.1"
dependencies = [
  "rdkit-pypi",
  "tqdm",
  "numpy>=1.21.0",
  "click>=8.0.1"
]
```

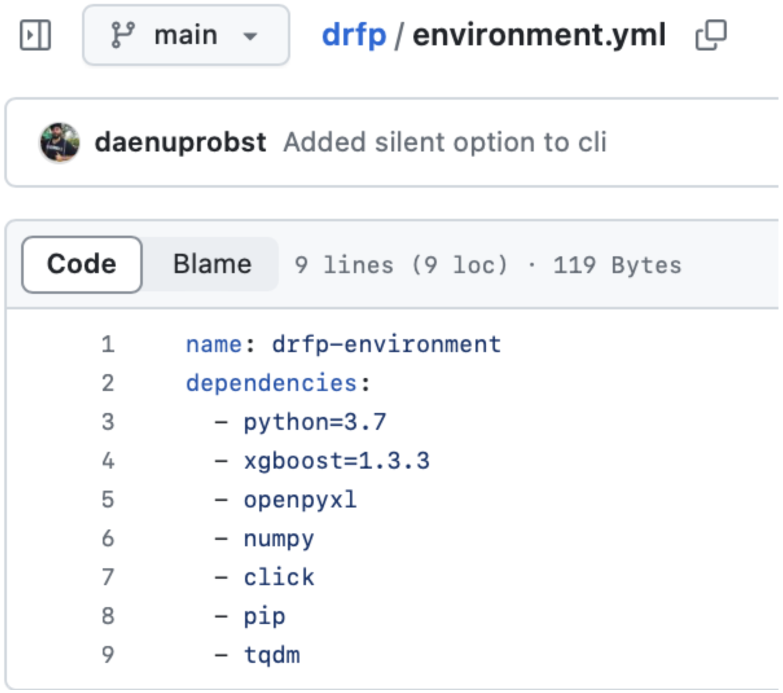
"pip install ."

- The setup.py (old option), setup.py + setup.cfg (old option) or **pyproject.toml** (recommended) make it possible to use pip to install your package.
- It will recognise **subfolders containing an "__init__.py"** file as part of the package. This file can be empty, but it must exist.
- pip install . (“.” means the current folder) will
  - Identify the package (finds the pyproject.toml file)
  - Build the package (using the build tool defined in the pyproject.toml)
  - Resolves dependencies (installs all the required package dependencies)
  - Then it installs the package, by copying the package files into the *site-packages directory* of the environment (this is where all the installed packages are)
- To uninstall a package, you run ”pip uninstall package-name”

"pip install -e ." (when you are working on a package)

- If you are working on a package, and doing changes to it. You would like the changes to be reflected in your code directly without having to reinstall the package.
- That's what the "-e" flag (editable) is for.
- Instead of installing the package in the site-packages directory. It links the installation to the current directory.
- Hence, if you change something in the code, it will immediately change in the installed package.
- → recommended when you are working no a package

environment.yml



daenuprobst Added silent option to cli

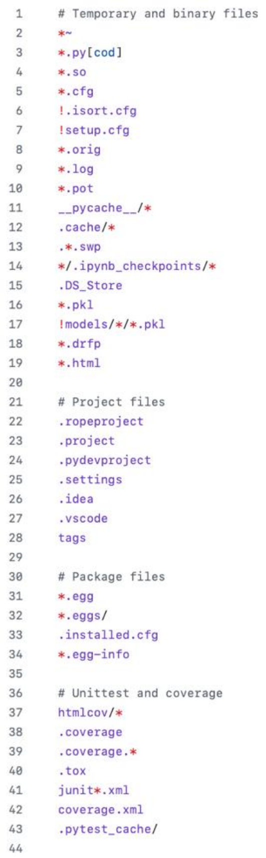
9 lines (9 loc) · 119 Bytes

```
name: drfp-environment
dependencies:
  - python=3.7
  - xgboost=1.3.3
  - openpyxl
  - numpy
  - click
  - pip
  - tqdm
```

File to create a **conda environment**

- Might contain some additional packages which are installed with conda instead of pip
- Other packages that are required to reproduce example results, but not dependencies of the package.

.gitignore

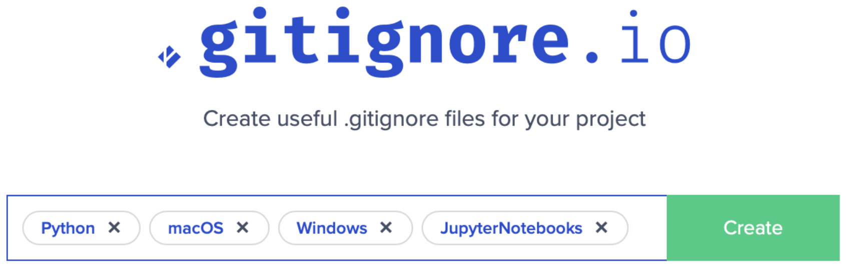


```
# Temporary and binary files
*~
*.py[cod]
*.so
*.cfg
!.isort.cfg
!setup.cfg
*.orig
*.log
*.pot
__pycache__/*
.cache/*
.*.swp
*/.ipynb_checkpoints/*
.DS_Store
*.pkl
!models/*/*.pkl
*.drfp
*.html

# Project files
.ropeproject
.project
.pydevproject
.settings
.idea
.vscode
tags

# Package files
*.egg
*.eggs/
.installed.cfg
*.egg-info

# Unittest and coverage
htmlcov/*
.coverage
.coverage.*
.tox
junit*.xml
coverage.xml
.pytest_cache/

```

Whatever matches what is specified in the .gitignore, will be ignored by git.

How do I know what to include?

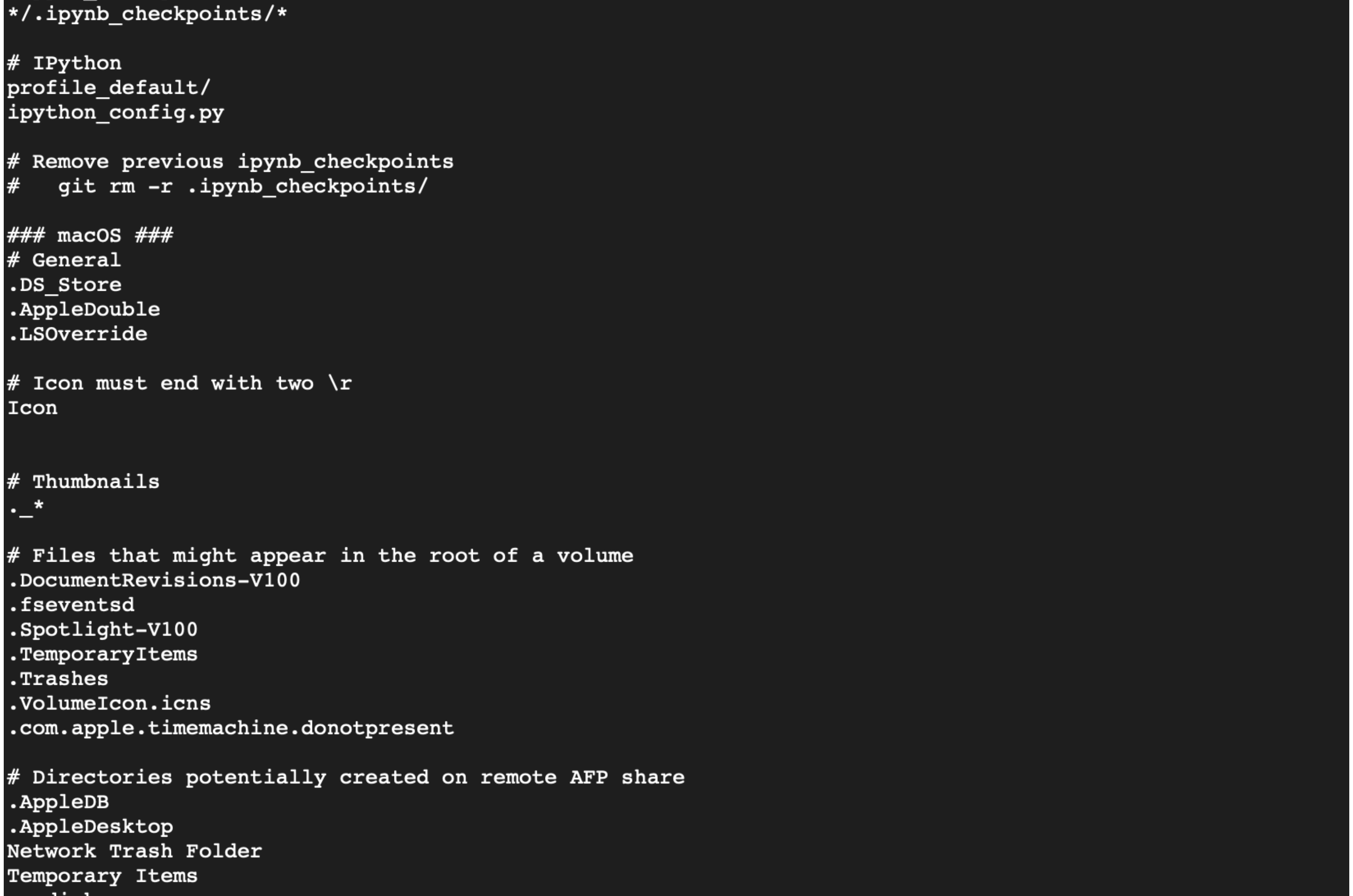
I typically start from https://gitignore.io or the one suggested on GitHub.



gitignore.io

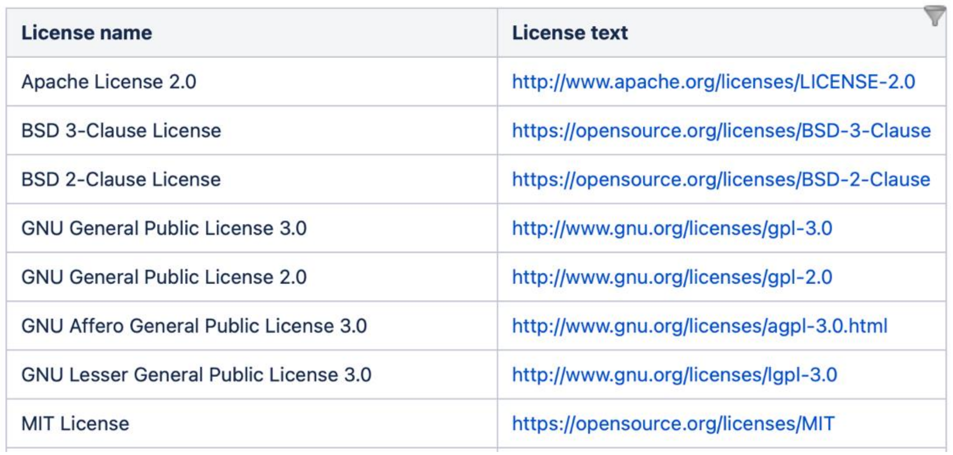
Create useful .gitignore files for your project

https://www.toptal.com/developers/gitignore/api/python,macos,windows,jupyternotebooks



```
*/.ipynb_checkpoints/*

# IPython
profile_default/
ipython_config.py

# Remove previous ipynb_checkpoints
#   git rm -r .ipynb_checkpoints/

### macOS ###
# General
.DS_Store
.AppleDouble
.LSOverride

# Icon must end with two \r
Icon


# Thumbnails
._*

# Files that might appear in the root of a volume
.DocumentRevisions-V100
.fseventsd
.Spotlight-V100
.TemporaryItems
.Trashes
.VolumeIcon.icns
.com.apple.timemachine.donotpresent

# Directories potentially created on remote AFP share
.AppleDB
.AppleDesktop
Network Trash Folder
Temporary Items
```

Code licenses



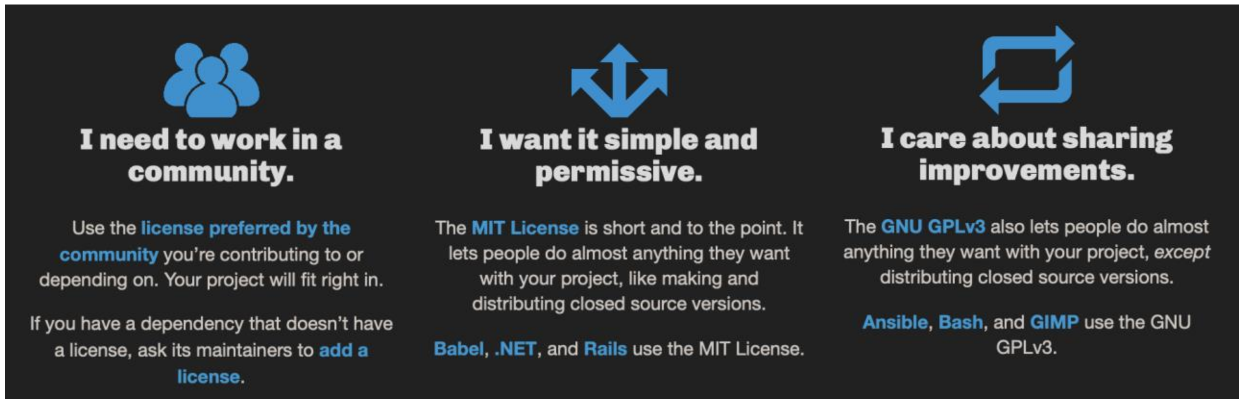
License name	License text
Apache License 2.0	http://www.apache.org/licenses/LICENSE-2.0
BSD 3-Clause License	https://opensource.org/licenses/BSD-3-Clause
BSD 2-Clause License	https://opensource.org/licenses/BSD-2-Clause
GNU General Public License 3.0	http://www.gnu.org/licenses/gpl-3.0
GNU General Public License 2.0	http://www.gnu.org/licenses/gpl-2.0
GNU Affero General Public License 3.0	http://www.gnu.org/licenses/agpl-3.0.html
GNU Lesser General Public License 3.0	http://www.gnu.org/licenses/lgpl-3.0
MIT License	https://opensource.org/licenses/MIT

Permissive

Viral licenses, forces people to republish modified code under same license (plenty of companies/labs will not touch that code)

Simple and permissive

https://choosealicense.com



I need to work in a community.

Use the license preferred by the community you're contributing to or depending on. Your project will fit right in.

If you have a dependency that doesn't have a license, ask its maintainers to add a license.

I want it simple and permissive.

The MIT License is short and to the point. It lets people do almost anything they want with your project, like making and distributing closed source versions.

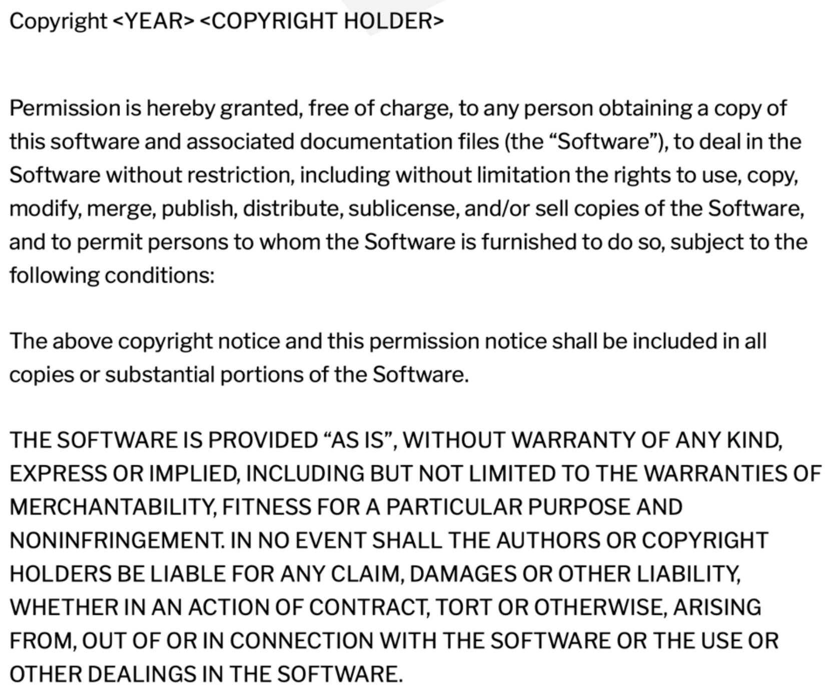
Babel, .NET, and Rails use the MIT License.

I care about sharing improvements.

The GNU GPLv3 also lets people do almost anything they want with your project, *except* distributing closed source versions.

Ansible, Bash, and GIMP use the GNU GPLv3.

MIT License

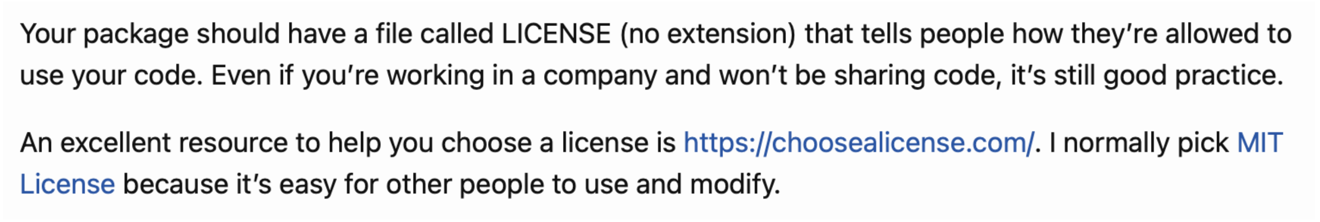


Copyright <YEAR> <COPYRIGHT HOLDER>

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

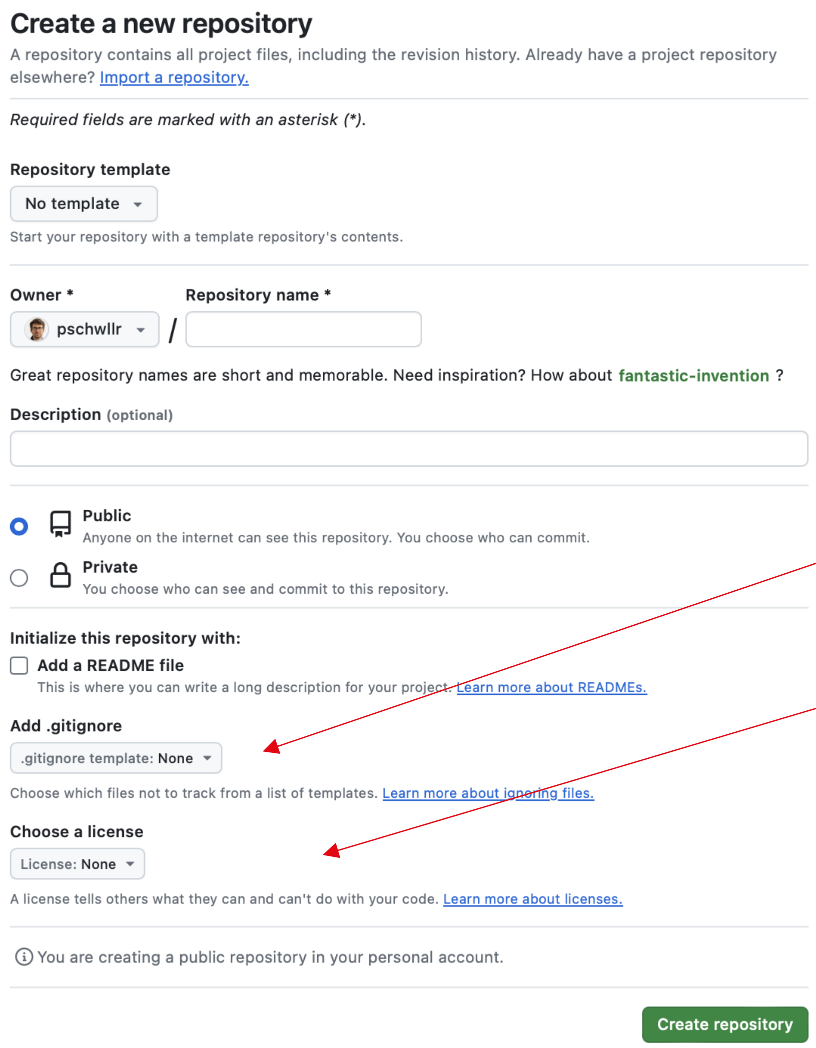
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.



Your package should have a file called LICENSE (no extension) that tells people how they're allowed to use your code. Even if you're working in a company and won't be sharing code, it's still good practice.

An excellent resource to help you choose a license is https://choosealicense.com/. I normally pick MIT License because it's easy for other people to use and modify.

https://cthoyt.com/2020/06/03/how-to-code-with-me-organization



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Required fields are marked with an asterisk (*).

Repository template

No template

Start your repository with a template repository's contents.

Owner * pschwllr / **Repository name ***

Great repository names are short and memorable. Need inspiration? How about fantastic-invention ?

Description (optional)

[✓ **Public**] Anyone on the internet can see this repository. You choose who can commit.

Private You choose who can see and commit to this repository.

Initialize this repository with:

☐ **Add a README file**
This is where you can write a long description for your project. Learn more about READMEs.

Add .gitignore

.gitignore template: None

Choose which files not to track from a list of templates. Learn more about ignoring files.

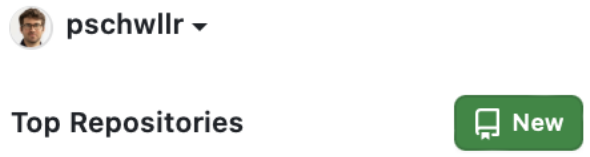
Choose a license

License: None

A license tells others what they can and can't do with your code. Learn more about licenses.

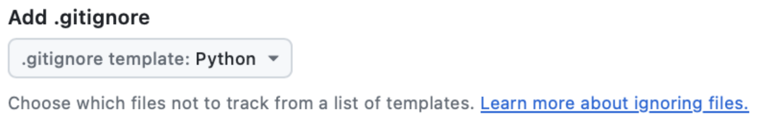
ⓘ You are creating a public repository in your personal account.

Create repository



pschwllr

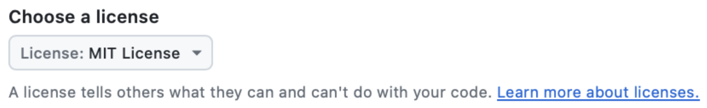
Top Repositories New



Add .gitignore

.gitignore template: Python

Choose which files not to track from a list of templates. Learn more about ignoring files.



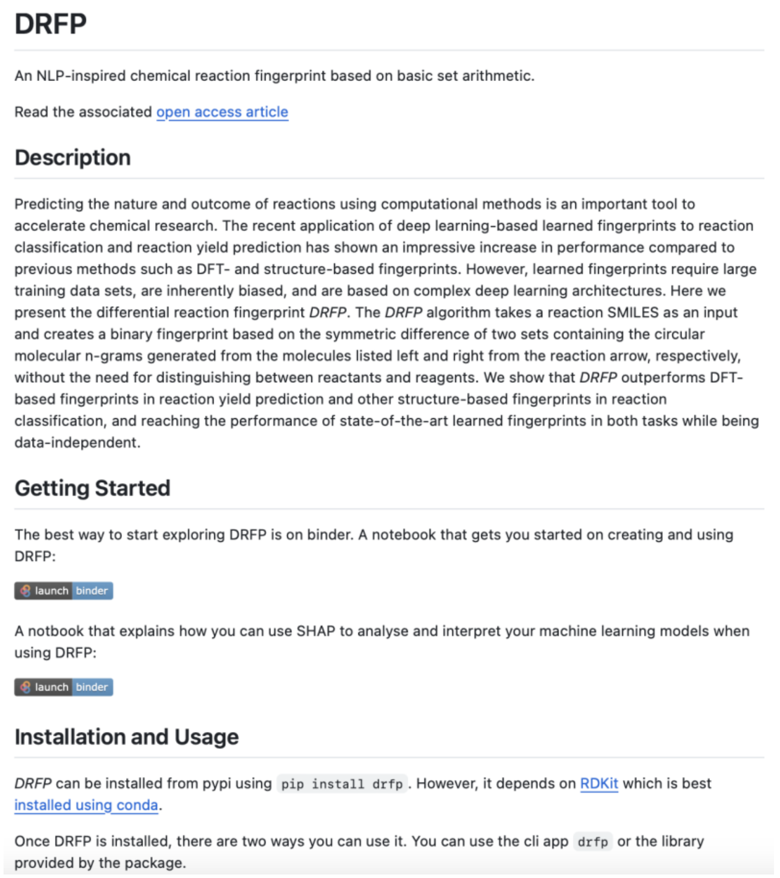
Choose a license

License: MIT License

A license tells others what they can and can't do with your code. Learn more about licenses.

README.md

- Nice introduction and description of your package
- Describe how to install it
- Make examples of main functionality



DRFP

An NLP-inspired chemical reaction fingerprint based on basic set arithmetic.

Read the associated open access article

Description

Predicting the nature and outcome of reactions using computational methods is an important tool to accelerate chemical research. The recent application of deep learning-based learned fingerprints to reaction classification and reaction yield prediction has shown an impressive increase in performance compared to previous methods such as DFT- and structure-based fingerprints. However, learned fingerprints require large training data sets, are inherently biased, and are based on complex deep learning architectures. Here we present the differential reaction fingerprint *DRFP*. The *DRFP* algorithm takes a reaction SMILES as an input and creates a binary fingerprint based on the symmetric difference of two sets containing the circular molecular n-grams generated from the molecules listed left and right from the reaction arrow, respectively, without the need for distinguishing between reactants and reagents. We show that *DRFP* outperforms DFT-based fingerprints in reaction yield prediction and other structure-based fingerprints in reaction classification, and reaching the performance of state-of-the-art learned fingerprints in both tasks while being data-independent.

Getting Started

The best way to start exploring DRFP is on binder. A notebook that gets you started on creating and using DRFP:

launch binder

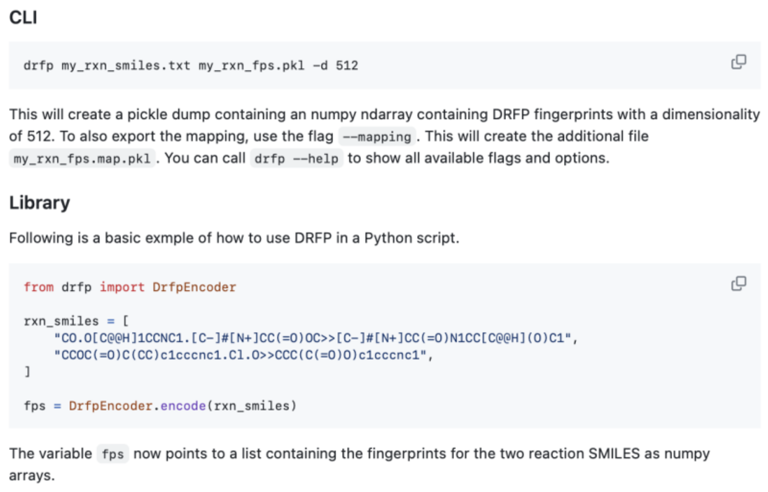
A notbook that explains how you can use SHAP to analyse and interpret your machine learning models when using DRFP:

launch binder

Installation and Usage

DRFP can be installed from pypi using `pip install drfp`. However, it depends on RDKit which is best installed using conda.

Once DRFP is installed, there are two ways you can use it. You can use the cli app `drfp` or the library provided by the package.



CLI

```
drfp my_rxn_smiles.txt my_rxn_fps.pkl -d 512
```

This will create a pickle dump containing an numpy ndarray containing DRFP fingerprints with a dimensionality of 512. To also export the mapping, use the flag `--mapping`. This will create the additional file `my_rxn_fps.map.pkl`. You can call `drfp --help` to show all available flags and options.

Library

Following is a basic exmple of how to use DRFP in a Python script.

```
from drfp import DrfpEncoder

rxn_smiles = [
    "CO.O[C@@H]1CCNC1.[C-]#[N+]CC(=O)OC>>[C-]#[N+]CC(=O)N1CC[C@@H](O)C1",
    "CCOC(=O)C(CC)c1cccnc1.Cl.O>>CCC(C(=O)O)c1cccnc1",
]

fps = DrfpEncoder.encode(rxn_smiles)
```

The variable `fps` now points to a list containing the fingerprints for the two reaction SMILES as numpy arrays.

How to create the complex package/project structure automatically?

- Pyscaffold (https://pyscaffold.org/en/stable/)
- Cookiecutter (https://cookiecutter.readthedocs.io/en/1.7.2/usage.html)
- Copier (https://copier.readthedocs.io/en/stable/)

- Example of a Cookiecutter template from my lab: https://github.com/schwallergroup/liac-repo

- We might move to a simpler Copier template in the future.

Minimal package

https://github.com/schwallergroup/copier-liac-minimal

```
├── .gitignore     # from gitignore.io or GitHub
├── LICENSE.txt     # e.g. MIT
├── README.md                    # Your package landing page
├── data                         # Folder to place data
├── notebooks            # Folder to place .ipynb files
├── pyproject.toml       # Package build configuration
├── scripts              # Auxiliary .py files
├── src
|   └── my_ch200_project                  # the package
|       └── __init__.py          # required (can be empty)
|       └── module_01.py                  # give descriptive names
|       └── module_02.py
└── tests                        # Folder with tests for package
```

```
copier copy gh:schwallergroup/copier-liac-minimal ch200_project
```

Going from scripts/notebooks to building packages gives you Python superpowers!