Molecular descriptors and Fingerprints calculation

Álvaro Román Gómez

5/29/23

Table of contents


```
# PATH TO CUSTOM MODULES
import sys

sys.path.append("../src")

# IMPORT LIBRARIES
import pandas as pd
import numpy as np

# IMPORT CUSTOM MODULES
import utils.moleculesUtils as moleculesUtils
from utils.moleculesUtils import *
```

```
# DIRECTORIES
input_path = "../data/raw/"
output_path = "../data/raw/"
# FILES
input_file = "CHEMBL4523954_raw.csv"
```

```
# LOAD DATA
molecules = pd.read_csv(input_path + input_file)
```

0.1 DATASET DESCRIPTION

A first descption of the dataset is obtained where the number of rows and columns is shown, as well as the names of the columns. Also, the data types of the columns will be shown:

The dataset has

```
print(
    "The dataset has "
    + str(molecules.shape[0])
    + " rows and "
    + str(molecules.shape[1])
    + " columns."
)
```

```
The dataset has 299 rows and 45 columns.
```

```
# DATASET COLUMN NAMES
print("The columns of the dataset are:")
print(molecules.columns)
```

```
The columns of the dataset are:
Index(['activity_comment', 'activity_id', 'activity_properties',
       'assay_chembl_id', 'assay_description', 'assay_type',
       'assay_variant_accession', 'assay_variant_mutation', 'bao_endpoint',
       'bao_format', 'bao_label', 'canonical_smiles', 'data_validity_comment',
       'data_validity_description', 'document_chembl_id', 'document_journal',
       'document_year', 'ligand_efficiency', 'molecule_chembl_id',
       'molecule_pref_name', 'parent_molecule_chembl_id', 'pchembl_value',
```

```
       'potential_duplicate', 'qudt_units', 'record_id', 'relation', 'src_id',
       'standard_flag', 'standard_relation', 'standard_text_value',
       'standard_type', 'standard_units', 'standard_upper_value',
       'standard_value', 'target_chembl_id', 'target_organism',
       'target_pref_name', 'target_tax_id', 'text_value', 'toid', 'type',
       'units', 'uo_units', 'upper_value', 'value'],
      dtype='object')
```

```
# NUMBER OF COLUMNS PER DATA TYPE
print("The dataset has:")
print(molecules.dtypes.value_counts())
```

```
The dataset has:
object     27
float64    11
int64       7
dtype: int64
```

```
# COLUMNS THAT HAVE ALL UNIQUE VALUES
print("The columns that have all unique values are:")
print(molecules.columns[molecules.nunique() == molecules.shape[0]])
```

```
The columns that have all unique values are:
Index(['activity_id'], dtype='object')
```

```
# COLUMNS THAT HAVE THE SAME VALUE
print("The columns that have the same value are:")
print(molecules.columns[molecules.nunique() == 1])
```

```
The columns that have the same value are:
Index(['assay_type', 'bao_endpoint', 'data_validity_comment',
       'data_validity_description', 'qudt_units', 'src_id', 'standard_flag',
       'standard_type', 'standard_units', 'target_chembl_id',
       'target_organism', 'target_pref_name', 'target_tax_id', 'type',
       'uo_units'],
      dtype='object')
```

0.2 REMOVE DUPLICATES

```
# REMOVE DUPLICATES
molecules = molecules.drop_duplicates(subset=["molecule_chembl_id"], ignore_index=True)
```

0.3 REMOVE UNNECESSARY COLUMNS

```
# REMOVE UNNECESSARY COLUMNS
molecules = molecules[["canonical_smiles", "standard_value"]]
```

0.4 DEFINE ACTIVE AND INACTIVE MOLECULES

```
# CALCULATE STANDARD VALUE MEDIAN
standard_value_median = molecules["standard_value"].median()
```

```
# DEFINE ACTIVE AND INACTIVE MOLECULES
molecules["activity"] = np.where(
    molecules["standard_value"] < standard_value_median, 1, 0
)
# COUNT ACTIVE AND INACTIVE MOLECULES
print("The dataset has:")
print(str(molecules["activity"].value_counts()[1]) + " active molecules")
print(str(molecules["activity"].value_counts()[0]) + " inactive molecules")
```

```
The dataset has:
141 active molecules
143 inactive molecules
```

1 CALCULATE MOLECULAR DESCRIPTORS AND FINGERPRINTS

According to the literature, several moelcular descriptors and fingerprints have been used in the construction of models to predict the activity of molecules. The selection of these descriptors and fingerprints plays a key role in model performance. Sometimes, even more important than the model itself.

After literature review, the following molecular descriptors and fingerprints were selected to be computed and therefore used in the models construction:

- General and topological molecular descriptors (RDKit)
- MACCS keys (MACCS)
- Extended connectivity fingerprints (ECFP)

1.1 MOLECULAR DESCRIPTORS

```
# CALCULATE MOLECULAR DESCRIPTORS
molecular_descriptors = moleculesUtils.calculate_molecule_set(
    smiles_set=molecules,
    smiles_column="canonical_smiles",
    function=calculate_molecular_descriptors,
)
```

1.2 FINGERPRINTS

```
# CALCULATE MACC KEYS
macc_keys = moleculesUtils.calculate_molecule_set(
    smiles_set=molecules,
    smiles_column="canonical_smiles",
    function=calculate_maccs_keys,
)
```

```
# CALCULATE ECFP FINGERPRINTS
ecfp4_fingerprints = moleculesUtils.calculate_molecule_set(
    molecules, "canonical_smiles", calculate_ecfp4_fingerprints
)
```

2 SAVE DATA

The data obtained from the calculation of molecular descriptors and fingerprints will be saved in a csv file called `chembl_data.csv`.

In total, there will be one dataframe for each type of molecular descriptor and fingerprint. Each of them will count with the SMILES string of the molecules and the corresponding standard

value that represents the activity of the molecules. The SMILES can be seen as ID of the molecules and it will be removed in further analysis.

```
# REMOVE SMILES AND STANDARD VALUE COLUMNS FROM ALL DATAFRAMES
molecules = molecules.drop(["standard_value"], axis=1)
molecular_descriptors = molecular_descriptors.drop(
    ["canonical_smiles", "standard_value"], axis=1
)
macc_keys = macc_keys.drop(["canonical_smiles", "standard_value"], axis=1)
ecfp4_fingerprints = ecfp4_fingerprints.drop(
    ["canonical_smiles", "standard_value"], axis=1
)
```

```
# SAVE CHEMBL DATA
molecules.to_csv(output_path + "smiles_activity.csv", index=False)
# SAVE MOLECULAR DESCRIPTORS
molecular_descriptors.to_csv(output_path + "molecular_descriptors.csv", index=False)
# SAVE MACCS KEYS
macc_keys.to_csv(output_path + "macc_keys.csv", index=False)
# SAVE ECFP FINGERPRINTS
ecfp4_fingerprints.to_csv(output_path + "ecfp4_fingerprints.csv", index=False)
```