Guide to Creating a Molecular HuggingFace Dataset

HuggingFace is a platform for sharing Datasets, Models, and Spaces. This tutorial describes best practices for creating a dataset for molecular data. With a dataset in-hand and no prior experience, it could date 30-minutes to a few couple of hours to create and upload a dataset.

1 Identify dataset to create

Navigate to Rosetta Data Bazaar Curation Sheet

- Find a dataset to curate or add one
- Good choices for datasets
    - large, high-quality, and of broad interest to the Rosetta Community and beyond
    - Not well curated by other curation efforts (e.g. deep mutational data curation through https://www.mavedb.org/)
    - license compatible with sharing, if in doubt, ask!

2 Getting Started with HuggingFace

- Create a HuggingFace account
- Navigate to the Rosetta Commons HuggingFace Organization page

HuggingFace documentation resources

- HuggingFace Dataset guide
- Dataset Card Documentation
- Dataset Card Specification
- Dataset Card Template

Gating access to dataset

- Gated datasets

Dataset Card

The dataset card is a standardized description of the dataset that facilitates its use and limitations. For context, (Mitchell et al. 2018) provide guidance for the related concept of Model Cards.

> Model Cards for Model Reporting
> Margaret Mitchell, Simone Wu, Andrew Zaldivar, Parker Barnes, Lucy Vasserman, Ben Hutchinson, Elena Spitzer, Inioluwa Deborah Raji, Timnit Gebru (2018) https://arxiv.org/abs/1810.03993

Some aspects of the dataset card that are specific to Rosetta Data Bazaar Datasets:

3 Create Dataset

Set up Personal Access Tokens (PAT)

See the help page on how to set up security tokens. This is needed to clone/push the repository using git

- Navigate to: https://huggingface.co/settings/tokens
- Click Create New Token → fill out information
- Save the token, e.g. in a password manager

Data processing workflow overview

1. Create pilot datasets in personal space and then once ready transfer to the Rosetta Data Bazaar collection
    a. Click Name icon ⇒ New → Dataset
        i. Fill out model name,
    b. Navigate to "Files and Versions" → README.md
    c. Fill out the top Dataset Card metadata (you can come back and fill out more details later)
2. Web-workflow
    a. Edit READ.md directly in the browser
    b. upload/delete other files directly
3. Git workflow Clone the repo to your computer
    a. `git clone https://<user_name>:<security_token>@huggingface.co/<repo_path>`
    b. create analysis folder structure

```
src/            # scripts for data curation
data/           # stored raw data for processing/curation
intermediate/   # store processed/curated data for uploading
```

    c. Add `.gitignore`

```
data/*
intermediate/*
```

    d. Use standard git workflow for modifying README.md and curation scripts

Uploading data to HuggingFace

There are range ways to structure dataset organization, getting the `push_to_hub()` right is a little tricky

Here is a working example from the MegaScale dataset.

1. Process data through scripts and store each split as a separate `.parquet` file

```
intermediate/<dataset_id>_<split_id>.parquet
```

2. Use datasets package to load the local dataset into memory. See below for more examples of how to load different types of datasets

```
dataset_tag = "dataset3"
cache_dir = "path/to/scratch"
```

```
dataset = datasets.load_dataset(
    "parquet",
    name = dataset_tag,
    data_dir = "./intermediate",
    data_files = {
        "train" : f"{dataset_tag}_train.parquet",
        "validation" : f"{dataset_tag}_valdation.parquet",
        "test" : f"{dataset_tag}_test.parquet"},
    cache_dir = cache_dir,
    keep_in_memory = True)
```

3. Set up Personal Access Keys on HuggingFace (see above)

4. Use the datasets package to push the dataset to hub, for ex
```
repo_id = "RosettaCommons/MegaScale"
dataset.push_to_hub(
    repo_id = repo_id,
    config_name = dataset_tag,
    data_dir = f"{dataset_tag}/data",
    commit_message = "Upload {dataset_tag}")
```

5. This produces on HuggingFace
```
https://huggingface.co/datasets/{repo_id}/tree/main/{dataset_tag}/data/
      {split}-<split-chunk-index>-of-<n-split-chunks>.parquet
```

Downloading data from HuggingFace

1. To load the dataset, optionally select specific split and/or columns
```
dataset = datasets.load_dataset(
        path = repo_id,
        name = dataset_tag,
        data_dir = dataset_tag,
        cache_dir = cache_dir,
        keep_in_memory = True)
```

2. If needed, convert data to pandas
```
import pandas as pd
df = dataset.data['train'].to_pandas()
```

4 Add an informative README.md

The `README.md` is a markdown file that is displayed when goes to the front page for the dataset. It should give appropriate context for the dataset and guidance on how to use it. As a template, consider having these sections--where teh parts in brackets should be filled in. See the MIP dataset as an example.

```
# <DATASET TITLE>
<short descriptive abstract the dataset>


## Quickstart Usage

### Install HuggingFace Datasets package

Each subset can be loaded into python using the HuggingFace
[datasets](https://huggingface.co/docs/datasets/index) library.
First, from the command line install the `datasets` library

    $ pip install datasets

Optionally set the cache directory, e.g.

    $ HF_HOME=${HOME}/.cache/huggingface/
    $ export HF_HOME

then, from within python load the datasets library

    >>> import datasets

### Load model datasets

To load one of the <DATASET ID> model datasets, use `datasets.load_dataset(...)`:

    >>> dataset_tag = "<DATASET TAG>"
    >>> dataset_model = datasets.load_dataset(
      path = "<HF PATH TO DATASET>",
      name = f"{dataset_tag}_models",
      data_dir = f"{dataset_tag}_models")['train']


and the dataset is loaded as a `datasets.arrow_dataset.Dataset`

    >>> dataset_models
    <RESULT OF LOADING DATASET MODEL>

which is a column oriented format that can be accessed directly, converted in to a
`pandas.DataFrame`, or `parquet` format, e.g.

    >>> dataset_models.data.column('<COLUMN NAME IN DATASET>)
    >>> dataset_models.to_pandas()
    >>> dataset_models.to_parquet("dataset.parquet")

### <BREIF EXAMPLE OF HOW TO USE DIFFERENT PARTS OF THE DATASET>
```

```
## Dataset Details

### Dataset Description
<DETAILED DESCRIPTION OF DATASET>


- **Acknowledgements:**
<ACKNOWLEDGEMENTS>


- **License:** <LICENSE>

### Dataset Sources
- **Repository:** <URL FOR SOURCE OF DATA>
- **Paper:**
  <APA CITATION REFERENCE FOR SOURCE DATA>
- **Zenodo Repository:** <ZENODO LINK IF RELEVANT>


## Uses
<DESCRIPTION OF INTENDED USE OF DATASET>

### Out-of-Scope Use
<DESCRIPTION OF OUT OF SCOPE USES OF DATASET>


### Source Data
<DESCRIPTION OF SOURCE DATA>

## Citation
<BIBTEX REFERENCE FOR DATASET>

## Dataset Card Authors
<NAME/INFO OF DATASET AUTHORS>
```

5 Add Metadata to the Dataset Card

Overview

A the top of the `README.md file include metadata about the dataset in yaml format

```
---
language: ...
license: ...
size_categories: ...
```

```
pretty_name: '...'
tags: ...
dataset_summary: ...
dataset_description: ...
acknowledgements: ...
repo: ...
citation_bibtex: ...
citation_apa: ...
---
```

For the full spec, see the Dataset Card specification

- Dataset Card Documentation
- Dataset Card Specification
- Dataset Card Template

To allow the datasets to be loaded automatically through the datasets python library, additional info needs to be in the header of the README.md. It should reflect how the repository is structured

```
configs:
dataset_info:
```

While it is possible to create these by hand, it highly recommended allowing it to be created automatically when uploaded via loading the dataset locally with datasets.load_dataset(...), then pushing it to the hub with datasets.push_to_hub(...)

- Example of uploading data using push_to_hub()
- See below for more details about how to use push_to_hub(...) for different common formats

Metadata fields

License

- If the dataset is licensed under an existing standard license, then use it
- If it is unclear, then the authors need to be contacted for clarification
- Licensing it under the Rosetta License
    - Add the following to the dataset card:

```
license: other
license_name: rosetta-license-1.0
license_link: LICENSE.md
```

    - Upload the Rosetta LICENSE.md to the Dataset

Citation

- If the dataset has a DOI (e.g. associated with a published paper), use doi2bib.org
- DOI → APA converter:

tags

- Standard tags for searching for HuggingFace datasets
- typically:

```
- biology
- chemistry
```

repo

- Github, repository, figshare, etc. URL for data or project

citation_bibtex

- Citation in bibtex format
- You can use https://www.doi2bib.org/

citation_apa

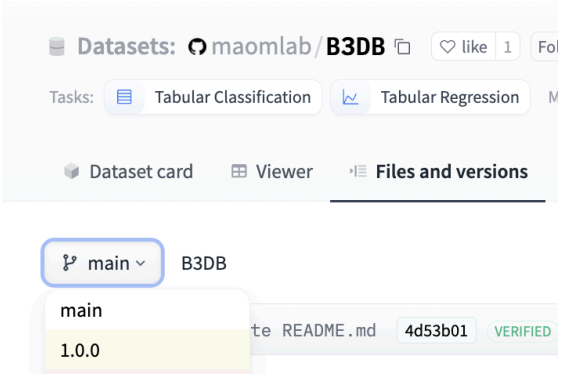
- Citation in APA format

6 Tag the version of the curation

Once you have made a dataset public, create a tagged branch using the `create_tag` function that allows for including the dataset in reproducible workflows. Versions typically follow standard semantic versioning guidelines of `<major>.<minor>.<patch>`.

```
import huggingface_hub
huggingface_hub.create_tag(
      repo_id = "<org_name>/<dataset_name>",
      tag = "<version>",
      tag_message = "<message>",
      repo_type = "dataset")
```

To access the tagged branch from the web-interface, go to the dataset → Files and version → select from the drop-down menu the tagged version:



To download a specific tagged branch for a dataset, add the `revision="<version>"` argument to `load_dataset()`:

```
import datasets
dataset = datasets.load_dataset("<org_name>/<dataset_name>", revision="1.0")
```

Dataset curation practical recommendations

Structure of data in a HuggingFace datasets

Datasets, sub-datasets, splits

- A HuggingFace dataset contains multiple sub-datasets e.g. at different filter/stringency levels.
- Each sub-dataset has one or more splits, typically ('train', 'validate', 'test'). If the data does not have splits it will be 'train'.
- The data in different splits of a single sub-dataset should non-overlapping
- Example:
    - The MegaScale contains 6 datasets
        - dataset1 # all stability measurements
        - dataset2 # high-quality folding stabilities
        - dataset3 # ΔG measurements
        - dataset3_single # ΔG measurements of single-point mutants with ThermoMPNN (Dieckhaus, et al., 2024) splits
        - dataset3_single_cv # 5-fold cross validation of ΔG measurements of single-point mutants with ThermoMPNN (Dieckhaus, et al., 2024) splits
    - To load a specific subdataset:
        - `datasets.load_dataset(path = "RosettaCommons/MegaScale", name = "dataset1", data_dir = "dataset1")`

Example: One .csv file dataset

One table named `outcomes.csv` to be pushed to HuggingFace dataset repository `maomlab/example_dataset`

First load the dataset locally then push it to the hub:

```
import datasets
dataset = datasets.load_dataset(
    "csv",
    data_files ="outcomes.csv",
    keep_in_memory = True)

dataset.push_to_hub(repo_id = "maomlab/example_dataset")
```

This will create the following files in the repo

```
data/
      train-00000-of-00001.parquet
```

and add the following to the header of README.md

```
dataset_info:
  features:
    - name: id
      dtype: int64
    - name: value
      dtype: int64
  splits:
    - name: train
      num_bytes: 64
      num_examples: 4
  download_size: 1332
  dataset_size: 64
configs:
  - config_name: default
    data_files:
      - split: train
        path: data/train-*
```

to load these data from HuggingFace

```
dataset = datasets.load_dataset("maomlab/example_dataset")
```

Example: train/valid/test split .csv files

Three tables train.csv, valid.csv, test.csv to be pushed to HuggingFace dataset repository `maomlab/example_dataset`
load the three splits into one dataset and push it to the hub:

```
import datasets
dataset = datasets.load_dataset(
      'csv',
      data_dir = "/tmp",
      data_files = {
            'train': 'train.csv',
            'valid': 'valid.csv',
            'test': 'test.csv'},
      keep_in_memory = True)

dataset.push_to_hub(repo_id = "maomlab/example_dataset")
```

This will create the following files in the repo

```
data/
```

```
train-00000-of-00001.parquet
valid-00000-of-00001.parquet
test-00000-of-00001.parquet
```

and add the following to the header of the README.md

```
dataset_info:
  features:
    - name: id
      dtype: int64
    - name: value
      dtype: int64
  splits:
    - name: train
      num_bytes: 64
      num_examples: 4
    - name: valid
      num_bytes: 64
      num_examples: 4
    - name: test
      num_bytes: 64
      num_examples: 4
  download_size: 3996
  dataset_size: 192
configs:
  - config_name: default
    data_files:
      - split: train
        path: data/train-*
      - split: valid
        path: data/valid-*
      - split: test
        path: data/test-*
```

to load these data from HuggingFace

```
dataset = datasets.load_dataset("maomlab/example_dataset")
```

Example: sub-datasets

If you have different related datasets (`dataset1.csv`, `dataset2.csv`, `dataset3.csv`) that should go into a single repository but contain different types of data so they aren't just splits of the same dataset, then load each dataset separately and push it to the hub with a given config name.

```
import datasets
dataset1 = datasets.load_dataset('csv', data_files = '/tmp/dataset1.csv',
keep_in_memory = True)
dataset2 = datasets.load_dataset('csv', data_files = '/tmp/dataset2.csv',
keep_in_memory = True)
dataset3 = datasets.load_dataset('csv', data_files = '/tmp/dataset3.csv',
keep_in_memory = True)
```

```
dataset1.push_to_hub(repo_id = "maomlab/example_dataset", config_name =
'dataset1', data_dir = 'dataset1/data')
dataset2.push_to_hub(repo_id = "maomlab/example_dataset", config_name =
'dataset2', data_dir = 'dataset2/data')
dataset3.push_to_hub(repo_id = "maomlab/example_dataset", config_name =
'dataset3', data_dir = 'dataset3/data')
```

This will create the following files in the repo

```
dataset1/
      data/
            train-00000-of-00001.parquet
dataset2/
      data/
            train-00000-of-00001.parquet
dataset3/
      data/
            train-00000-of-00001.parquet
```

and add the following to the header of the README.md

```
dataset_info:
  - config_name: dataset1
    features:
      - name: id
        dtype: int64
      - name: value1
        dtype: int64
    splits:
      - name: train
        num_bytes: 64
        num_examples: 4
    download_size: 1344
    dataset_size: 64
  - config_name: dataset2
    features:
      - name: id
        dtype: int64
      - name: value2
        dtype: int64
    splits:
      - name: train
        num_bytes: 64
        num_examples: 4
    download_size: 1344
    dataset_size: 64
  - config_name: dataset3
    features:
      - name: id
        dtype: int64
      - name: value3
        dtype: int64
    splits:
```

```
      - name: train
        num_bytes: 64
        num_examples: 4
    download_size: 1344
    dataset_size: 64
configs:
  - config_name: dataset1
    data_files:
      - split: train
        path: dataset1/data/train-*
  - config_name: dataset2
    data_files:
      - split: train
        path: dataset2/data/train-*
  - config_name: dataset3
    data_files:
      - split: train
        path: dataset3/data/train-*
```

to load these datasets from HuggingFace

```
dataset1 = datasets.load_dataset("maomlab/example_dataset", name = 'dataset1', data_dir = 'dataset1')
dataset2 = datasets.load_dataset("maomlab/example_dataset", name = 'dataset2', data_dir = 'dataset2')
dataset3 = datasets.load_dataset("maomlab/example_dataset", name = 'dataset3', data_dir = 'dataset3')
```

Format of a dataset

A dataset should consist of a single table where each row is a single observation
The columns should follow typical database design guidelines

- Identifier columns
    - sequential key
        - For example: [1, 2, 3, ...]
    - primary key
        - single column that uniquely identify each row
            - distinct for every row
            - no non-missing values
        - For example, for a dataset of protein structures from the Protein Data Bank, the PDB ID is the primary key
    - composite key
        - A set of columns that uniquely identify each row
            - Either hierarchical or complementary ids that characterize the observation
            - For example, for an observation of mutations, the (structure_id, residue_id, mutation_aa) is a unique identifier
    - additional/foreign key identifiers

        - identifiers to link the observation with other data
        - For example
            - for compounds identified by PubChem SubstanceID, the ZINC ID for the compound could be a foreign key
            - FDA drug name or the IUPAC substance name
- Tidy key/value columns
    - Tidy vs array data
        - tidy data sometimes called (long) has one measurement per row
            - Multiple columns can be used to give details for each measurement including type, units, metadata
            - Often good for certain data science computational analysis workflows (e.g. tidyverse/dplyr)
            - Can handle variable number of measurements per object
            - Duplicates object identifier columns for each measurement
        - array data sometimes called (wide) has one object per row and multiple measurements as different columns
            - Typically each measurement is typically a single column
            - More compact, i.e. no duplication of identifier columns
            - Good for certain ML/matrix based computational workflows

Molecular formats

- Store molecular structure in standard text formats
    - protein structure: PDB, mmCIF, modelCIF
    - small molecule: SMILES, InChi
    - use uncompressed, plaintext format
        - Easier to computationally analyze
        - the whole dataset will be compressed for data serialization
- Filtering / Standardization / sanitization
    - Be clear about process methods used to process the molecular data
    - Be especially careful for inferred / aspects of the data
        - protonation states,
        - salt form, stereochemistry for small molecules
        - data missingness including unstructured loops for proteins
    - Tools
        - MolVS is useful for small molecule sanitization

Computational data formats

- On disk formats
    - parquet format disk format
        - column oriented (so can load only data that is needed, easier to compress)

        - robust reader/write codes from apache arrow for Python, R etc.
    - ArrowTable
        - In memory format closely aligned with the on disk parquet format
        - Native format for datasets stored in datasets python package
    - tab/comma separated table
        - Prefer tab separated, more consistent parsing without needing escaping values
        - Widely used row-oriented text format for storing tabular data to disk
        - Does not store data format and often needs custom format conversion code/QC for loading into python/R
        - Can be compressed on disk but row-oriented, so less compressible than .parquet
    - .pickle / .Rdata
        - language specific serialization of complex data structures
        - Often very fast to read/write, but may not be robust for across language/OS versions
        - Not easily interoperable across programming languages
- In memory formats
    - R data.frame/dplyr::tibble
        - Widely used format for R data science
        - Out of the box faster for tidyverse data manipulation, split-apply-combine workflows
    - Python pandas DataFrame
        - Widely used for python data science
        - Out of the box not super fast for data science
    - Python numpy array / R Matrix
        - Uses single data type for all data
        - Useful for efficient/matrix manipulation
    - Python Pytorch dataset
        - Format specifically geared for loading data for Pytorch deep-learning

Recommendations

- On disk
    - For small, config level tables use .tsv
    - For large data format use .parquet
        - Smaller than .csv/.tsv
        - Robust open source libraries in major language can read and write .parquet files faster than .csv/.tsv
- In memory
    - Use dplyr::tibble / pandas DataFrame for data science tables
    - Use numpy array / pytorch dataset for machine learning

Questions and Answers

My dataset is a metadata curation of multiple datasets from different places, how should I license it?

If you are building on others' work, It is important to respect their licenses. How you do this will fall into three buckets

- The data cannot be used, e.g. because of a proprietary license restriction
- The data can be used with or without some restriction. For example, if the source dataset is licensed under the creative commons open source license CC BY-SA 4.0, it requires redistribution to "ShareAlike". Or, the authors require signing a specific usage license, for example, the Rocklin lab requires registering the use of the MegaScale dataset so they can demonstrate impact to maintain grant support.
- The license of the source data is not clear. In this case, it is best to reach out to the original authors and either request that they adopt a license (open source or otherwise), or get explicit permission to re-share the data

My dataset is made up of a bunch of small tabular datasets, should I make them each a sub-dataset or different "splits"?