AIQC

Team AIQC

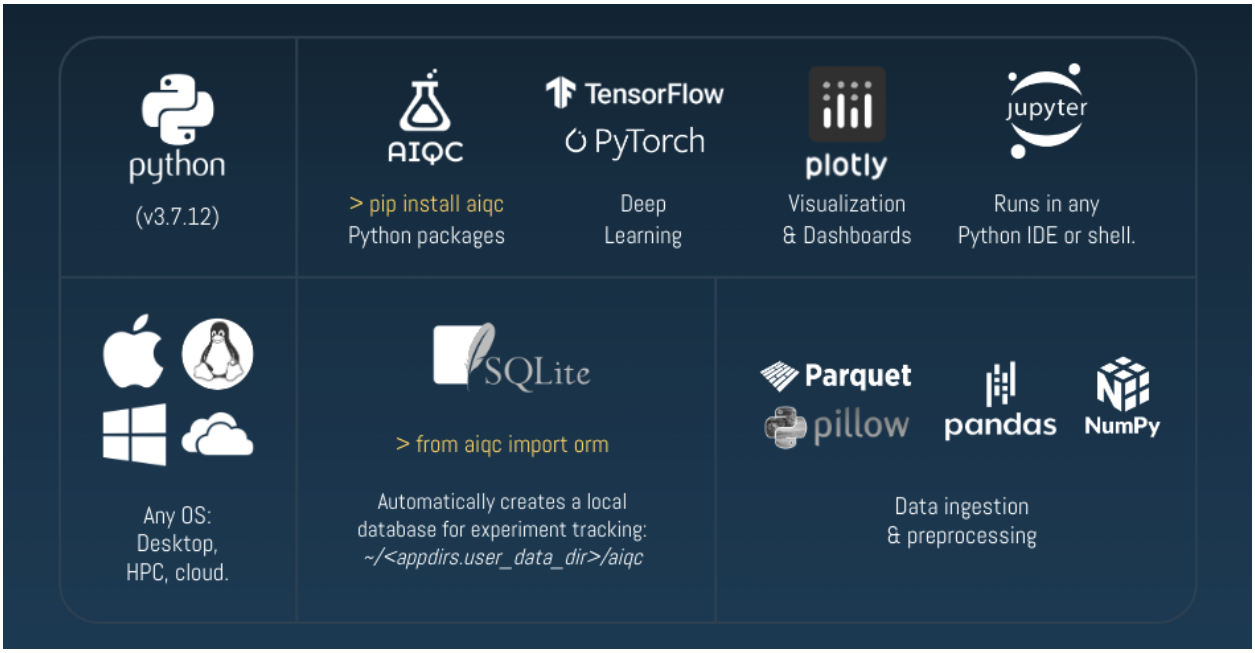
Aug 09, 2023

GETTING STARTED

CHAPTER

ONE

INSTALL



1.1 AIQC Python Package

[ ]:
```
pip install --upgrade pip
pip install --upgrade wheel
pip install --upgrade aiqc
```

If during troubleshooting you find yourself reinstalling unwanted packages from the cache, then use:

```
pip install --upgrade --no-cache-dir aiqc
```

If that doesn't work, read the rest of this notebook (e.g. supported Python versions).

1.2 Environment Setup

AIQC has many dependencies with specific versions, so we recommend creating a new virtual environment that is solely dedicated to AIQC using either PyEnv or Conda.

1.2.1 Python Version

Requires Python 3+ (check your deep learning library's Python requirements). AIQC was developed on Python 3.7.12 in order to ensure compatibility with Google Colab.

Conda does not provide 3.7.12, but AIQC has been tested on 3.7.16 as well so you can use that version.

Additionally, check the Python version required by the machine learning libraries that you intend to use. For example, at the time this was written, Tensorflow/ Keras required Python 3.5–3.8. If you need more information about dependencies, the PyPI `setup.py` is in the root of the github.com/aiqc/aiqc repository.

[1]:
```
import sys
sys.version
```

[1]:
```
'3.7.12 (default, Dec 10 2021, 10:49:04) \n[Clang 13.0.0 (clang-1300.0.29.3)]'
```

Pickle Disclaimer

AIQC, much like PyTorch, relies heavily on Pickle for saving Python objects in its database. Therefore, as a caveat of Pickle, if you create objects in your `aiqc.sqlite` file using one version of Python and try to interact with it on a newer version of Python, then you may find that pickle is no longer able to deserialize the object. For this reason, `sys.version` and other helpful info about your OS/ Python version is stored in the `config.json` file at the time of creation.

1.2.2 Operating System

AIQC was designed to be OS-agnostic. It has been tested on the following operating systems:

- macOS 10.15 and 11.6.1
- Linux (Ubuntu, Alpine, RHEL).
- Windows 10 (and WSL).

> If you run into trouble with the installation process on your OS, please create a GitHub discussion so that we can attempt to resolve, document, and release a fix as quickly as possible.

1.2.3 Optional - JupyterLab IDE

AIQC runs anywhere Python runs. We just like Jupyter for interactive visualization and data transformation. FYI, *jupyterlab* is not an official dependency of the AIQC package.

[ ]:
```
pip install jupyterlab
```

JupyterLab requires Node.js >= 10. Once all extensions switch to JupyterLab 3.0 prebuilding, this will no longer be necessary.

[4]:
```
!node -v
```

```
v14.7.0
```

1.2.4 Optional - Swap Space for Failover Memory

On local machines, it is good practice to configure "swap space." This way, if your processes run out of memory/ RAM, then the excess information will simply spill over onto the (potentially dynamically sized) swap partition of your hard drive, as opposed to causing an out-of-memory crash. For GB sized datasets, spinning media HDDs (5,400/ 7,200 RPM) may be too slow for usage with swap, but you could get by with NVMe/ SSD.

1.3 Location of AIQC Files

AIQC makes use of the Python package, `appdirs`, for an operating system (OS) agnostic location to store configuration and database files. This not only keeps your `$HOME` directory clean, but also helps prevent careless users from deleting your database.

> The installation process checks not only that the corresponding appdirs folder exists on your system but also that you have the permissions neceessary to read from and write to that location. If these conditions are not met, then you will be provided instructions during the installation about how to create the folder and/ or grant yourself the appropriate permissions.
>
> We have attempted to support both Windows (`icacls` permissions and backslashes `C:\\`) as well as POSIX including Mac and Linux including containers & Google Colab (`chmod letters` permissions and slashes `/`). Note: due to variations in the ordering of appdirs author and app directories in different OS', we do not make use of the appdirs `appauthor` directory, only the `appname` directory.

1.3.1 Location Based on OS

Test it for yourself:

```
import appdirs; appdirs.user_data_dir('aiqc');
```

- Mac: `/Users/Username/Library/Application Support/aiqc`
- Linux - Alpine and Ubuntu: `/root/.local/share/aiqc`
- Windows: `C:\Users\Username\AppData\Local\aiqc`

1.3.2 Database

The database is simply a SQLite **file**, and AIQC serves as an ORM/ API for that SQL database.

> So you ***do not*** have to worry about anything like installing a database server, database client, database users, configuring ports, configuring passwords/ secrets/ environment variables, or starting and restopping the database. Shoutout to the ORM, peewee. Glad we found this fantastic and simple alternative to SQLAlchemy.

1.3.3 Config

The configuration file contains low level information about: * Where AIQC should persist data. * Runtime (Python, OS) environment for reproducibility and troubleshooting.

1.4 Optional - Deleting the Database

If, for whatever reason, you find that you need to destroy your SQLite database file and start from scratch, then you can do so without having to manually find and `rm` the database file. In order to reduce the chance of an accident, `confirm:bool=False` by default.

> Bear in mind that if you are on either a server or shared OS, then this database may contain more than just your data.

1.4.1 a) One-Liner

Both `confirm:bool=False` and `rebuild:bool=False`, so it only does what you command it to do.

[ ]:
```
from aiqc.orm import create_db, destroy_db
```

[4]:
```
destroy_db(confirm=True, rebuild=True)
```

```
=> Success - deleted database file at path:
/Users/layne/Library/Application Support/aiqc/aiqc.sqlite3


=> Success - created database file at path:
/Users/layne/Library/Application Support/aiqc/aiqc.sqlite3


  Success - created all database tables.
```

1.4.2 b) Or Line-by-Line

[5]:
```
destroy_db(confirm=True)
```

```
=> Success - deleted database file at path:
/Users/layne/Library/Application Support/aiqc/aiqc.sqlite3
```

[6]:
```
create_db()
```

```
=> Success - created database file at path:
/Users/layne/Library/Application Support/aiqc/aiqc.sqlite3
```

(continues on next page)

(continued from previous page)

```
Success - created all database tables.
```

1.5 Troubleshooting

1.5.1 Reloading the Package

After CRUD'ing the config files, AIQC needs the be reimported in order to detect those changes. This can be done in one of three ways:

- If everything goes smoothly, it should automatically happen behind the scenes: `reload(sys.modules['aiqc'])`.
- Manually by the user: `from importlib import reload; reload(aiqc)`.
- Manually restarting your Python kernel/ session and `import aiqc`.

CHAPTER

TWO

UI

AIQC makes comparing and evaluating models effortless with its reactive Dash-Plotly user interface. The following dashboards put precalculated metrics & charts for each split/fold of every model right at your fingertips.

> Reference the Evaluation section for more information about the plots and metrics.

2.1 Experiment Tracker

During the training process, practitioners continually improve their algorithm by experimenting with different combinations of architectures and parameters. This iterative process generates a lot of post-processing data, and it's difficult to figure out which model is the best just by staring at hundreds of rows of raw data.

2.2 Compare Models Head-to-Head

The head-to-head comparison provides a deep dive that helps tease out the answers to challenging questions:

> How does a practitioner know that 'model A' is actually better than 'model B' for their use case? Is one model slightly more biased than the other? What characteristics in the data is each model relying on? Can we get higher performance if we train for just a bit longer?

2.3 What-If Analysis

Ever wonder "What if?" By providing a dynamic user inferface for inference, AIQC allows you to tweak the inputs for a scenario in order to simulate its outcome.

Its applications are endless: Will the patient *survive* if their blood pressure drops? Will this drug be *effective* with 1 more rotational bond? Will the gene editing *increase* CO2 sequestration?

- By default, the feature inputs are populated with either the median numeric/ mode categoric value depending on their dtype. Metadata about the feature's distribution can be seen by hovering over the column name.
- If feature importance was enabled during model evaluation, then the feature columns are presented in rank-order of median feature importance (as seen in the first row of the hover tooltip).
- The inputs are pre/post-processed via *aiqc.mlops.Inference* using the original model's *aiqc.mlops.Pipeline*.

- Clicking the star uses `BaseModel.flip_star()` to toggle `Prediction.is_starred` as a favorite indicator.
- Right now this page is only configured for supervised analysis (regression, binary classification, multi-label classification) on tabular data. However, this foundation can easily be extended to support the other AIQC data/analysis combinations.

2.4 Run the App

The app must be launched from the command line as a Python module.

```
$ python -m aiqc.ui.app

Dash is running on http://127.0.0.1:9991/

 * Running on http://127.0.0.1:9991 (Press CTRL+C to quit)
```

If you attempt to terminate the server with `CTRL+Z` by accident, then the port will get hung. The freeport package makes it easy to release the port in this case.

The `--port int` and `--debug` mode are configurable.

```
$ python -m aiqc.ui.app --help

    usage: aiqc.ui.app [-h] [--port] [--debug] [--no-debug]

    Launch AIQC's Dash-Plotly UI for experiment tracking
    https://dash.plotly.com/devtools

    optional arguments:
      -h, --help  show this help message and exit
      --port      localhost:<port> to run on. Default=9991
      --debug     Raises errors and inspects callbacks.
      --no-debug  By default, neither raises errors nor inspects callbacks.
```

The page refreshes every 10 seconds.

> If, for some reason, you find that your queries are taking longer than 10 seconds to finish, please start a discussion: https://github.com/aiqc/AIQC/discussions

---

2.5 What about JupyterDash?

Initially, the UI was built around `jupyter_dash`, which enabled running the Dash app within either a JupyterLab cell or tab. However, this approach was not stable for the following reasons:

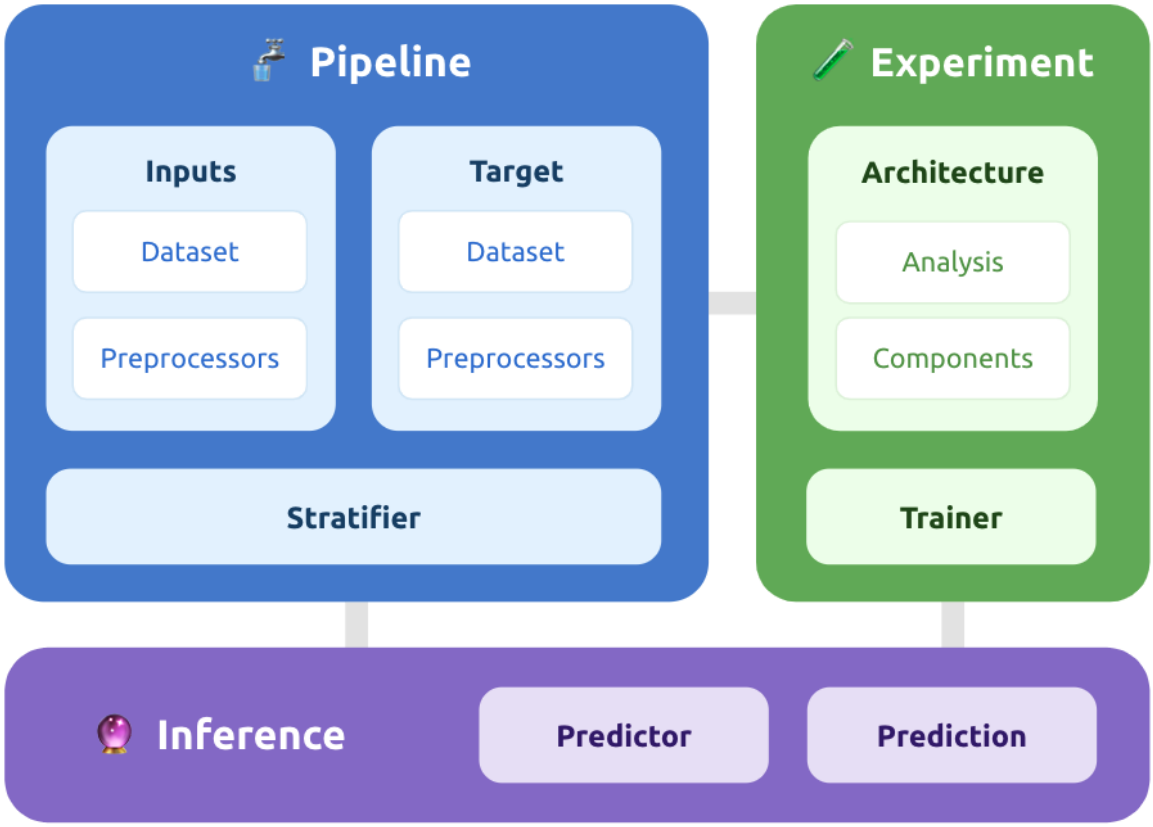
- Hung & unkillable ports
- When `_terminate_server_for_port` was removed in v0.4.2, it became unusable.
- Werkzeug deprecation warnings

JupyterLab ships with a terminal. So technically the app can still be launched from within the JupyterLab user interface without resorting to Pythonic `sys` commands.

CHAPTER

THREE

API



3.1 Declarative

The *High-Level API* is *declarative*. What does that mean? All you have to do is specify the *state* that you want the data in, and then the backend executes all of the tedious data wrangling needed to achieve that state. It's like Terraform for machine learning.

```
from aiqc.orm import Dataset
from aiqc.mlops import *
```

1. `Pipeline` declares how to preprocess data.

2. `Experiment` declares variations of models to train and evaluate.
3. `Inference` declares new samples to predict.

Reference the tutorials to the see the high level API in action for various types of data and analysis. It's declarative nature makes it easy to learn by reading examples as opposed to piecing together which arguments point to each other. Check back here if you get stuck.

> *Why so many pointer variables?* – Under the hood, the High-Level API is actually chaining together a workflow using the *object-relational model (ORM)* of the Low-Level API. Many of the classes provided here are just an easier-to-use versions of their ORM counterparts.

3.2 1. `Pipeline`

Declares how to prepare data. The steps defined within the pipeline are used at multiple points in the machine learning lifecycle:

- Preprocessing of training and evaluation data.
- Caching of preprocessed training and evaluation data.
- Post-processing (e.g. decoding) during model evaluation.
- Inference: encoding and decoding new data.

```
Pipeline(
    inputs
    , target
    , stratifier
    , name
    , description
)
```

Argument	Type	Default	Description
inputs	list(Input)	Required	*Input* - One or more featuresets
target	Target	None	*Target* - Leave blank during unsupervised/ self-supervised analysis.
stratifier	Stratifier	None	*Stratifier* - Leave blank during inference.
name	str	None	An auto-incrementing version will be assigned to Pipelines that share a name.
description	str	None	Describes how this particular workflow is unique.

> It is possible for an `Input` and a `Target` to share the same `Dataset`. The `Input.include_columns` and `Input.exclude_columns` will automatically be adjusted to exclude `Target.column`.

Returns	Splitset instance as seen in the Low-Level API. We will use this later as the *Trainer.pipeline* argument.

3.2.1 1a. Input

These are the features that our model will learn from.

This is a wrapper for `Feature` and all of its preprocessors in the Low-Level API.

```
Input(
    dataset
    , exclude_columns
    , include_columns
    , interpolaters
    , window
    , encoders
    , reshape_indices
)
```

Argument	Type	Default	Description
dataset	Dataset	Required	Dataset from Low-Level API
exclude_columns	list(str)	None	The columns from the Dataset that will *not* be used in the featureset
include_columns	list(str)	None	The columns from the Dataset that *will* be used in the featureset
interpolaters	list(Input.Interpolater)	None	*Input.Interpolater*
window	Input.Window	None	*Input.Window*
encoders	list(Input.Encoder)	None	*Input.Encoder*
reshape_indices	tuple(int/str/tuple)	None	Reference `FeatureShaper` from Low-Level API .

Both `exclude_columns` and `include_columns` cannot be used simultaneously.

1ai. Input.Interpolater

Used to fill in the blanks in a sequence.

This is a wrapper for `FeatureInterpolater` in the Low-Level API.

```
Input.Interpolater(
    process_separately
    , verbose
    , interpolate_kwargs
    , dtypes
    , columns
)
```

1aii. `Input.Window`

Used to slice and shift samples into many time series windows for walk-forward/ backward analysis.

This is a wrapper for `Window` in the Low-Level API.

```
Input.Window(
    size_window
    , size_shift
    , record_shifted
)
```

1aiii. `Input.Encoder`

Used to numerically encode data.

This is a wrapper for `FeatureCoder` in the Low-Level API.

```
Input.Encoder(
    sklearn_preprocess
    , verbose
    , include
    , dtypes
    , columns
)
```

3.2.2 1b. `Target`

What the model is trying to predict.

This is a wrapper for `Label` and all of its preprocessors in the Low-Level API.

```
Target(
    dataset
    , column
    , interpolater
    , encoder
)
```

Argument	Type	Default	Description
dataset	Dataset	Required	`Dataset` from Low-Level API
column	list(str)	None	The column from the Dataset to use as the target.
interpolater	Target.Interpolater	None	*Target.Interpolater*
encoder	Target.Encoder	None	*Target.Encoder*

1bi. Target.Interpolater

Used to fill in the blanks in a sequence.

This is a wrapper for `LabelInterpolater` in the Low-Level API.

```
Target.Interpolater(
    process_separately
    , interpolate_kwargs
)
```

1bii. Target.Encoder

Used to numerically encode data.

This is a wrapper for `LabelCoder` in the Low-Level API.

```
Target.Encoder(
    sklearn_preprocess
)
```

3.2.3 1c. Stratifier

Used to slice the dataset into training, validation, test, and/or cross-validated subsets.

This is a wrapper for `Splitset` in the Low-Level API.

```
Stratifier(
    size_test
    , size_validation
    , fold_count
    , bin_count
)
```

3.3 2. Experiment

Used to declare variations of models that will be trained.

```
Experiment(
    architecture
    , trainer
)
```

Argument	Type	Default	Description
architecture	Architecture	Required	*Architecture*
trainer	Trainer	Required	*Trainer*

Returns	Queue instance as seen in the Low-Level API.

3.3.1 2a. Architecture

The model and hyperparameters to be trained.

This is a wrapper for `Algorithm` in the Low-Level API, with the addition of hyperparameters.

```
Architecture(
    library
    , analysis_type
    , fn_build
    , fn_train
    , fn_optimize
    , fn_lose
    , fn_predict
    , hyperparameters
)
```

3.3.2 2b. Trainer

The options used for training.

This is a wrapper for `Queue` in the Low-Level API, with the addition of `pipeline`.

```
Trainer(
    pipeline
    , repeat_count
    , permute_count
    , search_count
    , search_percent
)
```

3.4 3. Inference

Used to preprocess new samples, run predictions on them, decode the output, and, optionally, evaluate the predictions.

```
Inference(
    predictor
    , input_datasets
    , target_dataset
    , record_shifted
)
```

Argument	Type	Default	Description
predictor	Predictor	Required	Predictor to use for inference
input_datasets	list(Dataset)	Required	New Datasets to run inference on.
target_dataset	Dataset	None	New Datasets for scoring inference. Leave this blank for pure inference where no metrics will be calculared.
record_shifted	bool	False	Set this to True for scoring during unsupervised time series inference

We don't need to specify fully-fledged `Inputs` and `Target` objects because the `Pipeline` of the `predictor` object will be reused in order to process these new datasets.

Returns	`Prediction` instance as seen in the Low-Level API.

CHAPTER

FOUR

ORM

4.1 Object-Relational Model

The Low-Level API is an *object-relational model* for machine learning. Each class in the ORM maps to a table in a SQLite database that serves as a machine learning *metastore*.

The real power lies in the relationships between these objects (e.g. `Label→Splitset←Feature` and `Queue→Job→Predictor→Prediction`), which enable us to construct rule-base protocols for various types of data and analysis.

Goobye, *X_train, y_test*. Hello, object-oriented machine learning.

```
from aiqc.orm import *
```

Automatic 'id' method argument

If an ORM-based classes is instantiated, then any method called by the resulting object will automatically pass in the object's `self.id` in as its first positional argument:

```
queue = Queue.get_by_id(id)
queue.run_jobs()
```

However, if the class has not been instantiated, then the `id` is required:

```
Queue.run_jobs(id)

Although I did not design this pattern, if you think about it, it makes sense. ORMs␣
↪allow you to fluidly traverse relational objects. If you had to check the ``object.
↪id`` of everything you returned before interacting with it, then that would ruin the␣
↪user-friendly experience.
```

4.2 0. BaseModel

The `BaseModel` class applies to all tables in the ORM. It's metadata in the truest sense of the word.

Localized timestamps are handled by utils.config.timezone_now(). They are made human-readable via `strftime('%Y%b%d_%H:%M:%S')` → "2022Jun23_07:13:14"

4.2.1 0a. Methods

└── `created_at()`

Returns the creation timestamp in human-readable format.

└── `updated_at()`

Returns timestamp of the most recent update in human-readable format.

└── `flip_star()`

A way to toggle (favorite/ unfavorite) the `is_starred` attribute in order to make entries easy to find.

└── `set_info()`

Add descriptive information about an entry so that you remember why you created it

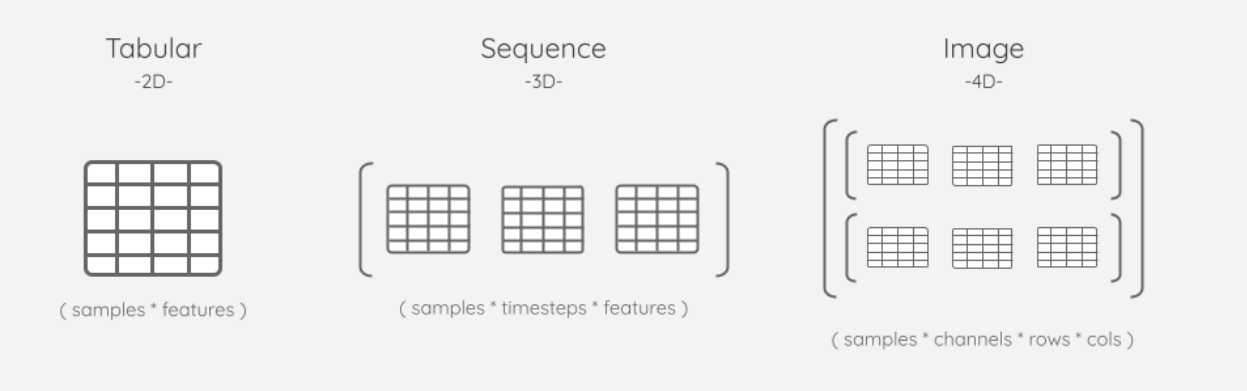
```
set_info(name, description)
```

Argument	Type	Default	Description
name	str	None	Short name to remember this entry by
description	str	None	What is unique about this entry?

4.2.2 0b. Attributes

Attribute	Type	Description
id	AutoField	Auto-incrementing integer (1-based, not zero-based) PrimaryKey
time_created	DateTime-Field	Records a timestamp when the record is created
time_updated	DateTime-Field	Records a timestamp when the record is created. Overwritten every time the record is updated.
is_starred	Boolean-Field	Used to indicated that the entry is a favorite
name	CharField	Short name to remember this entry by
description	CharField	What is unique about this entry?

4.3 1. Dataset



The `Dataset` class provides the following subclasses for working with different types of data:

Type	Dimensionality	Supported Formats	Format (if ingested)
Tabular	2D	Files (Parquet, CSV, TSV) / Pandas DataFrame (in-memory)	Parquet
Sequence	3D	NumPy (in-memory ndarray, npy file)	npy
Image	4D	NumPy (in-memory ndarray, npy file) / Pillow-supported formats	npy

The names are merely suggestive, as the primary purpose of these subclasses is to provide a way to register data of known dimensionality. For example, a practitioner could ingest many uni-channel/ grayscale images as a 3D Sequence Dataset instead of a multi-channel 4D Image Dataset.

Why not 2D NumPy? The `Dataset.Tabular` class is intended for strict, column-specific dtypes and Parquet persistence upon ingestion. In practice, this conflicted too often with NumPy's array-wide dtyping. We use the best tools for the job (df/pq for 2D) and (array/npy for ND).

4.3.1 1a. Methods

1ai. Registration

Most of the Dataset registration methods share these arguments/ concepts:

Argument	Description
ingest	Determines if raw data is either stored directly inside the metastore or remains on disk to be accessed via path/url. *In-memory* data like DataFrames and ndarrays must be ingested. Whereas *file-based* data like Parquet, NPY, Image folders/urls may remain remote. Regardless of whether or not the raw data is ingested, metadata is always derived from it by parsing: 2D via DataFrame and N-D via ndarray.
rename_columns	Useful for assigning column names to arrays or delimited files that would otherwise be unnamed. `len(rename_columns)` must match the number of columns in the raw data. Normally, an int-based range is assigned to unnamed columns. In this case, AIQC converts each column name to a string e.g. '1' during the registration process.
retype	Change the dtype of data using np.types. All Dataset subclasses support mass typing via `np.type`/ `str(np.type)`. Only the Tabular subclass supports inidividual column retyping via `dict(column=str(np.type))` ``. If ``rename_columns`` is used in conjuction with `retype=dict()`, then each `dict['column']` key must match its counterpart in rename_columns.
description	What information does this dataset contain? What is unique about this dataset/ version – did you edit the raw data, add rows, or change column names/ dtypes?
name	Triggers dataset *versioning*. Datasets that share a name will be assigned an auto-incrementing `version: int` number provided that they are not duplicates of each other based on a `sha256_hexdigest:str` hash. If you try to create an exact duplicate, it will warn you and `return` the matching duplicate instead of creating a new entity. This behavior makes it easy to rerun pipelines where Datasets are created inline.

Ingestion provides the following benefits, especially for entry-level users:

- Persist in-memory datasets (Pandas DataFrames, NumPy ndarrays).
- Keeps data coupled with the experiment in the portable SQLite file.
- Provides a more immutable and out-of-the-way storage location in comparison to a laptop file system.
- Encourages preserving tabular dtypes with the ecosystem-friendly Parquet format.

Why would I avoid ingestion?

- Happy with where the original data lives: e.g. S3 bucket.
- Don't want to duplicate the data.

  sha256? – It's the one-way hash algorithm that GitHub aspires to upgrade to. AIQC runs it on compressed data because it's easier and probably less-error prone than intercepting the bytes of the *fastparquet* intermediary tables before appending the Parquet magic bytes.

  Is SQLite a legitimate datastore? – In many cases, SQLite queries are faster than accessing data via a filesystem. It's a stable, 22 year-old technology that serves as the default database for iOS e.g. Apple Photos. AIQC uses it store raw data in byte format as a BlobField. I've stored tens-of-thousands of files in it over several years and never experienced corruption. Keep in mind that AWS S3 is blob store, and the Microsoft equivalent service is literally called Azure *Blob* Storage. The max size of a BlobField is 2GB, so ~20GB after compression. Either way, the goal of machine learning isn't to record the entire population within the weights of a neural network, it's to find subsets that are representative of the broader population.

1ai1. Dataset.Tabular

Here are some of the ways practitioners can use this 2D structure:

Multiple subjects (1 row per sample) * Multi-variate 1D (1 col per attribute)
Single subject (1 row per timestamp) * Multi-variate 1D (1 col per attribute)
Multiple subjects (1 row per timestamp) * Uni-variate 0D (1 col per sample)

Tabular datasets may contain both features and labels

└── Dataset.Tabular.from_df()

```
dataset = Dataset.Tabular.from_df(
    dataframe
    , rename_columns
    , retype
    , description
    , name
)
```

Argument	Type	Default	Description
df	DataFrame	Required	pd.DataFrame with int-based single index. DataFrames are always ingested.
rename_columns	list[str]	None	See Registration
retype	np.type / dict(column:np.type)	None	See Registration
description	str	None	See Registration
name	str	None	See Registration

└── Dataset.Tabular.from_path()

```
Dataset.Tabular.from_path(
    file_path
    , ingest
    , rename_columns
    , retype
    , header
    , description
    , name
)
```

Argument	Type	Default	Description
file_path	str	Required	Parsed based on how the file name ends (.parquet, .tsv, .csv)
ingest	bool	True	See Registration. Defaults to True because I don't want to rely on CSV files as a source of truth for dtypes, and compression works great in Parquet.
rename_columns	list[str]	None	See Registration
retype	np.type / dict(column:np.type)	None	See Registration
header	object	None	See Registration
description	str	None	See Registration
name	str	None	See Registration

1ai2. `Dataset.Sequence`

Here are some of the ways practitioners can use this 3D structure:

Single subject (1 patient) * Multiple 2D sequences
Multiple subjects * Single 2D sequence

> Sequence datasets are somewhat multi-modal in that, in order to perform supervised learning on them, they must eventually be paired with a `Dataset.Tabular` that acts as its `Label`.

└── `Dataset.Sequence.from_numpy()`

```
Dataset.Sequence.from_numpy(
    arr3D_or_npyPath
    , ingest
    , rename_columns
    , retype
    , description
    , name
)
```

Argument	Type	Default	Description
arr3D_or_npyPath	object / str	Required	3D array in the form of either an ndarray or npy file path
ingest	bool	None	See Registration. If left blank, ndarrays will be ingested and npy will not. Errors if ndarray and False.
rename_columns	list[str]	None	See Registration
retype	np.type / dict(column:np.type)	None	See Registration
description	str	None	See Registration
name	str	None	See Registration

1ai3. Dataset.Image

Here are some of the ways you can practitioners this 4D structure:

Single subject (1 patient) * Multiple 3D images
Multiple subjects * Single 3D image

Users can ingest 4D data using either: - The Pillow library, which supports various formats - Or NumPy arrays as a simple alternative

> Image datasets are somewhat multi-modal in that, in order to perform supervised learning on them, they must eventually be paired with a `Dataset.Tabular` that acts as its `Label`.

└── `Dataset.Image.from_numpy()`

```
Dataset.Image.from_numpy(
    arr4D_or_npyPath
    , ingest
    , rename_columns
    , retype
    , description
    , name
)
```

Argument	Type	Default	Description
arr4D_or_npyPath	ndarray / str	Required	4D array in the form of either an ndarray or npy file path
ingest	bool	None	See Registration. If left blank, ndarrays will be ingested and npy will not. Errors if input is ndarray and `ingest==False`.
rename_columns	list[str]	None	See Registration
retype	np.type / dict(column:np.type)	None	See Registration
description	str	None	See Registration
name	str	None	See Registration

└── `Dataset.Image.from_folder()`

```
Dataset.Image.from_folder(
    folder_path
    , ingest
    , rename_columns
    , retype
    , description
```

(continues on next page)

(continued from previous page)

```
    , name
)
```

Argument	Type	Default	Description
folder_path	str	Required	Folder of images to be ingested via Pillow. All images must be cropped to the same dimensions ahead of time.
ingest	bool	False	See Registration
rename_columns	list[str]	None	See Registration
retype	np.type / dict(column:np.type)	None	See Registration
description	str	None	See Registration
name	str	None	See Registration

└── `Dataset.Image.from_urls()`

```
Dataset.Image.from_urls(
    urls
    , source_path
    , ingest
    , rename_columns
    , retype
    , description
    , name
)
```

Argument	Type	Default	Description
urls	list(str)	Required	URLs that point to an image to be ingested via Pillow. All images must be cropped to the same dimensions ahead of time.
source_path	str	None	Optionally record a shared directory, bucket, or FTP site where images are stored. The backend won't use this information for anything.
ingest	bool	False	See Registration
rename_columns	list[str]	None	See Registration
retype	np.type / dict(column:np.type)	None	See Registration
description	str	None	See Registration
name	str	None	See Registration

1aii. Fetch

The following methods are exposed to end-users in case they want to inspect the data that they have ingested.

└── `Dataset.to_arr()`

Argument	Type	Default	Description
id	int	None	The identifier of the Dataset of interest
columns	list(str)	None	If left blank, includes all columns
samples	list(int)	None	If left blank, includes all samples

Subclass	Returns
Tabular	ndarray.ndim==2
Sequence	ndarray.ndim==3
Image	ndarray.ndim==4

└── `Dataset.to_df()`

Argument	Type	Default	Description
id	int	None	The identifier of the Dataset of interest
columns	list(str)	None	If left blank, includes all columns
samples	list(int)	None	If left blank, includes all samples

Subclass	Returns
Tabular	DataFrame
Sequence	list(DataFrame)
Image	list(list(DataFrame))

└── `Dataset.to_pillow()`

Argument	Type	Default	Description
id	int	None	The identifier of the Dataset of interest
samples	list(int)	None	If left blank, includes all samples

Subclass	Returns
Image	list(PIL.Image)

└── `Dataset.get_dtypes()`

Argument	Type	Default	Description
id	int	None	The identifier of the Dataset of interest
columns	list(str)	None	If left blank, includes all columns

Regardless of how the initial `Dataset.dtype` was formatted [e.g. single np.type / str(np.type) / dict(column=np.type)], this function intentionally returns then dtype of each column within a `dict(column=str(np.type)` format.

4.3.2 1b. Attributes

These are the fields in the Dataset table

Attribute	Type	Description
typ	CharField	The Dataset type: Tabular, Sequence, Image
source_format	CharField	The file format (Parquet, CSV, TSV) or in-memory class (DataFrame, ndarray)
source_path	CharField	The path of the original file/ folder
urls	JSONField	A list of URLs as an alternative to file paths/ folders
columns	JSONField	List of str-based names for each column
dtypes	JSONField	The type of each column. Tabular dtype is saved in `dict(column=str(np.type))` `` format. Where Sequence and Image dtype is saved in a singular ``str(np.type)`
shape	JSONField	Human-readable dictionary about the dimensions of the data e.g. `samples:10, columns:5`
sha256_hexdigest	CharField	A hash of the data to determine its uniqueness for versioning.
memory_MB	IntegerField	Size of the dataset in megabytes when loaded into memory
contains_nan	BooleanField	Whether or not the dataset contains any blank cells
header	PickleField	`pd.read_csv(header)` for TSV/CSV files.
is_ingested	BooleanField	Quick flag to see if the data was ingested. Exists to prevent querying the `blob` field unnecessarily.
blob	BlobField	The raw bytes of the data obtained via `BytesIO().getvalue`
version	IntegerField	The auto-incrementing version number assigned to unique datasets that share name

---

4.4 2. Feature

Determines the columns that will be used as predictive features during training. Columns is always the last dimension `shape[-1]` of a dataset.

4.4.1 2a. Methods

└── `Feature.from_dataset()`

```
Feature.from_dataset(
    dataset_id
    , include_columns
    , exclude_columns
)
```

Argument	Type	Default	Description
dataset_id	int	Required	`Dataset.id` from which you want to derive `Dataset.columns`.
include_columns	list(str)	None	Specify columns that *will* be included in the Feature. All columns that are not specified will *not* be included.
exclude_columns	list(str)	None	Specify columns that will *not* be included in the Feature. All columns that are not specified *will* be included.

If neither `include_columns` nor `exclude_columns` is defined, then all columns will be used.

Both `include_columns` and `exclude_columns` cannot be used at the same time

Fetch

Theses methods wrap Dataset's fetch methods:

Method	Arguments	Returns
to_arr()	id:int, columns:list(str)=Feature.columns, samples:list(int)=None	ndarray 2D / 3D / 4D
to_df()	id:int, columns:list(str)=Feature.columns, samples:list(int)=None	df / list(df) / list(list(df))
get_dtypes()	id:int, columns:list(str)=Feature.columns	dict(column=str(np.type))

4.4.2 2b. Attributes

These are the fields in the Feature table

Attribute	Type	Description
columns	JSONField	The columns included in this featureset
columns_excluded	JSONField	The columns, if any, in the dataset that were not included
fitted_featurecoders	PickleField	When `FeatureCoder`'s `fit` an sklearn preprocessor to these columns, the fit objects are saved here for downstream `inverse_transform`'ing
dataset	ForeignKeyField	Where these columns came from

4.5 3. Label

Determines the column(s) that will be used as a target during supervised analysis. Do no create a Label if you intend to conduct unsupervised/ self-supervised analysis.

4.5.1 3a. Methods

└── `Label.from_dataset()`

```
Label.from_dataset(
    dataset_id
    , columns
)
```

Argument	Type	Default	Description
dataset_id	int	Required	`Dataset.id` from which you want to derive `Dataset.columns`. Only Tabular Datasets may be used as a Label.
columns	list(str)	None	Specify columns that *will* be included in the Label. If left blank, defaults to all columns. If more than 1 column is provided, then the data in those columns must be in One-Hot Encoded (OHE) format.

Fetch

Theses methods wrap Dataset's fetch methods:

Method	Arguments	Returns
to_arr()	id:int, columns:list(str)=Label.col umns, samples:list(int)=None	ndarray 2D / 3D / 4D
to_df()	id:int, columns:list(str)=Label.col umns, samples:list(int)=None	df / list(df) / list(list(df))
get_dtypes()	id:int, columns:list(str)=Label.col umns	dict(column=str(np.type))

4.5.2 3b. Attributes

These are the fields in the Feature table

Attribute	Type	Description
columns	JSONField	The column(s) included in this featureset.
column_count	IntegerField	The number of columns in the Label. Used to determine if it is in validated OHE format or not
unique_classes	JSONField	Records all of the different values found in categorical columns. Not used for continuous columns.
fitted_labelcoder	PickleField	When a `LabelCoder` `fit`'s an sklearn preprocessor to these columns, the fit objects are saved here for downstream `inverse_transform`'ing
dataset	ForeignKeyField	Where these columns came from

4.6 4. Interpolate

If you don't have time series data then you do not need interpolation.

If you have continuous columns with missing data in a time series, then interpolation allows you to fill in those blanks mathematically. It does so by fitting a curve to each column. Therefore each column passed to an interpolater must satisfy: `np.issubdtype(dtype, np.floating)`.

Interpolation is the first preprocessor because you need to fill in blanks prior to encoding.

> `pandas.DataFrame.interpolate`
>
> https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.interpolate.html
>
> Is utilized due to its ease of use, variety of methods, and **support of sparse indices**. However, it does not follow the `fit/transform` pattern like many of the class-based sklearn preprocessors, so the interpolated training data is concatenated with the evalaution split during the interpolation of evaluation splits.

Below are the default settings if `interpolate_kwargs=None` that get passed to `df.interpolate()`. In my experience, `method=spline` produces the best results. However, if either (a) spline fails to fit to your data, or (b) you know that your pattern is linear - then try `method=linear`.

```
interpolate_kwargs = dict(
    method            = 'spline'
    , limit_direction = 'both'
    , limit_area      = None
    , axis            = 0
    , order           = 1
)
```

Because the sample dimension is different for each Dataset Type, they approach interpolation differently.

Dataset Type	Approach
Tabular	Unlike encoders, there is no `fit` object. So first the training data rows are interpolated independently. Then, when it comes time to interpolate other splits like validation, the training data is included in the sequence to be interpolated.
Sequence	Interpolation is ran on each 2D sequence separately
Image	Interpolation is ran on each 2D channel separately

4.6.1 4a. `LabelInterpolater`

Label is intended for a single column, so only 1 Interpolater will be used during `Label.preprocess()`

4ai. Methods

└── `LabelInterpolater.from_label()`

```
LabelInterpolater.from_label(
    label_id
    , process_separately
    , interpolate_kwargs
)
```

Argument	Type	Default	Description
label_id	int	Required	Points to the `Label.columns` to use
process_separately	bool	True	Used to restrict the fit to the training data, this may be flipped to `False`. However, doing so causes data leakage.
interpolate_kwargs	dict	None	Gets passed to `df.interpolate()`. See *Interpolate* section for defaults.

4aii. Attributes

These are the fields in the LabelInterpolater table

Attribute	Type	Description
process_separately	BooleanField	Whether or not the training data was interpolated by fitting to the entire dataset or not. Indicator of data leakage.
interpolate_kwargs	JSONField	Gets passed to `df.interpolate()`. See *Interpolate* section for defaults.
matching_columns	JSONField	The columns that were successfully interpolated
label	ForeignKeyField	The Label that this LabelInterpolater is applied to

---

4.6.2 4b. `FeatureInterpolater`

For *multivariate* datasets, columns/dtypes may need to be handled differently. So we use column/dtype filters to apply separate transformations. If the first transformation's filter includes a certain column/dtype, then subsequent filters may not include that column/dtype.

4bi. Methods

└── `FeatureInterpolater.from_feature()`

```
FeatureInterpolater.from_feature(
    feature_id
    , process_separately
    , interpolate_kwargs
    , dtypes
    , columns
    , verbose
)
```

Argument	Type	Default	Description
feature_id	int	Required	Points to the `Feature.columns` to use
process_separately	bool	True	Used to restrict the fit to the training data, this may be flipped to `False`. However, doing so causes data leakage.
interpolate_kwargs	dict	None	The `interpolate_kwargs:dict=N one` object is what gets passed to Pandas interpolation. In my experience, `method=spline` produces the best results. However, if either (a) spline fails to fit to your data, or (b) you know that your pattern is linear - then try `method=linear`.
dtypes	list(str)	None	The dtypes to include
columns	list(str)	None	The columns to include. Errors if any of the columns were already included by dtypes.
verbose	bool	True	If True, messages will be printed about the status of the interpolaters as they attempt to fit on the filtered columns

---

4bii. Attributes

These are the fields in the FeatureInterpolater table

Attribute	Type	Description
idx	IntegerField	Zero-based auto-incrementer that counts the number of FeatureInterpolaters attached to a Feature.
process_separately	BooleanField	Whether or not the training data was interpolated by fitting to the entire dataset or not. Indicator of data leakage.
interpolate_kwargs	JSONField	Gets passed to `df.interpolate()`. See *Interpolate* section for defaults.
matching_columns	JSONField	The columns that matched the filter
leftover_columns	JSONField	The columns that were not included in the filter
leftover_dtypes	JSONField	The dtypes that were not included in the filter
original_filter	JSONField	`dict().keys()==['include' ,'dtypes','columns']`
feature	ForeignKeyField	The Feature that this FeatureInterpolater is applied to

---

4.7 5. Encode

Transform data into numerical format that is close to zero. Reference Encoding for more information.

There are two phases of encoding: 1. `fit` on train - where the encoder learns about the values of the samples made available to it. Ideally, you only want to `fit` aka learn from your training split so that you are not leaking information from your validation and test spits into your model! However, categorical encoders are always fit on the entire dataset because they are not prone to leakage and any weights tied to empty OHE inputs will zero-out. 2. `transform` each split/fold

> Only sklearn.preprocessing methods are officially supported, but we have experimented with `sklearn.feature_extraction.text.CountVectorizer`

4.7.1 5a. `LabelCoder`

Label is intended for a single column, so only 1 LabelCoder will be used during `Label.preprocess()`

> Unfortunately, the name "LabelEncoder" is occupied by `sklearn.preprocessing.LabelEncoder`

5ai. Methods

└─ `LabelCoder.from_label()`

```
LabelCoder.from_label(
    label_id
    , sklearn_preprocess
)
```

Argument	Type	Default	Description
label_id	int	Required	Points to the `Label.columns` to use
sklearn_preprocess	object	Required	An instantiated `sklearn.preprocessing` class-based encoder - e.g. `StandardScaler()` neither `StandardScaler` nor `scale()`. AIQC will automatically correct the attributes of your encoder to smooth out any common errors they would cause. For example, preventing sparse SciPy matrix output (errors during tensor conversion) and data `copy()`.

5aii. Attributes

These are the fields in the LabelCoder table

Attribute	Type	Description
only_fit_train	BooleanField	Whether or not the encoder was fit on the training data or the entire dataset
is_categorical	BooleanField	If the encoder is meant for categorical data, and therefore automatically fit on the entire dataset
sklearn_preprocess	PickleField	The instantiated sklearn.preprocessing class that was fit
matching_columns	JSONField	The columns that matched the dtype/ column name filters
encoding_dimension	CharField	Did the encoder succeed on 1D/ 2D uni-column/ 2D multi-column?
label	ForeignKeyField	The Label that this LabelCoder is applied to

---

4.7.2 5b. `FeatureCoder`

For *multivariate* datasets, columns/dtypes may need to be handled differently. So we use column/dtype filters to apply separate transformations. If the first transformation's filter includes a certain column/dtype, then subsequent filters may not include that column/dtype.

5bi. Methods

└── `FeatureCoder.from_feature()`

```
FeatureCoder.from_feature(
    feature_id
    , sklearn_preprocess
    , include
    , dtypes
    , columns
    , verbose
)
```

Argument	Type	Default	Description
feature_id	int	Required	Points to the `Feature.columns` to use
sklearn_preprocess	object	Required	An instantiated `sklearn.preprocessing` class-based encoder - e.g. `StandardScaler()` neither `StandardScaler` nor `scale()`. AIQC will automatically correct the attributes of your encoder to smooth out any common errors they would cause. For example, preventing sparse SciPy matrix output (errors during tensor conversion) and data `copy()`.
include	bool	True	Whether to include or exclude the dtypes/columns that match the filter. You can create a filter for all columns by setting `include=False` and then setting both `dtypes` and `columns` to `None`.
dtypes	list(str)	None	The dtypes to filter
columns	list(str)	None	The columns to filter. Errors if any of the columns were already used by dtypes.
verbose	bool	True	If True, messages will be printed about the status of the encoders as they attempt to fit on the filtered columns

---

5bii. Attributes

These are the fields in the FeatureCoder table

Attribute	Type	Description
idx	IntegerField	Zero-based auto-incrementer that counts the number of FeatureCoders attached to a Feature.
sklearn_preprocess	PickleField	The instantiated sklearn.preprocessing class that was fit
encoded_column_names	JSONField	After the columns are encoded, what are their names? OHE appends `_<category>` to the original column names as it expands
matching_columns	JSONField	The columns that matched the filter
leftover_columns	JSONField	The columns that were not included in the filter
leftover_dtypes	JSONField	The dtypes that were not included in the filter
original_filter	JSONField	`dict().keys()==['include' ,'dtypes','columns']`
encoding_dimension	CharField	Did the encoder succeed on 1D/ 2D uni-column/ 2D multi-column?
only_fit_train	BooleanField	Whether or not the encoder was fit on the training data or the entire dataset
is_categorical	BooleanField	If the encoder is meant for categorical data, and therefore automatically fit on the entire dataset
feature	ForeignKeyField	The Feature that this FeatureCoder is applied to

4.8 6. Shape

Changes the shape of data. Only supports Features, not Labels.

Reshaping is applied at the end of `Feature.preprocess()`. So if the feature data has been altered via time series windowing or One Hot Encoder, then those changes will be reflected in the shape that is fed to `

When working with architectures that are highly dimensional such convolutional and recurrent networks (Conv1D, Conv2D, Conv3D / ConvLSTM1D, ConvLSTM2D, ConvLSTM3D), you'll often find yourself needing to reshape data to fit a layer's required input shape.

- *Reducing unused dimensions* - When working with grayscale images (1 channel, 25 rows, 25 columns) it's better to use Conv1D instead of Conv2D.
- *Adding wrapper dimensions* - Perhaps your data is a fit for ConvLSTM1D, but that layer is only supported in the nightly TensorFlow build so you want to add a wrapper dimension in order to use the production-ready ConvLSTM2D.

AIQC favors a *"channels_first"* (samples, channels, rows, columns) approach as opposed to *"channels_last"* (samples, rows, columns, channels).

> *Can't I just reshape the tensors during the training loop?* You could. However, AIQC systemtically provides the shape of features and labels to `Algorith.fn_build` to make designing the topology easier, so it's best to get the shape right beforehand. Additionally, if you reshape your data within the training loop, then you may also need to reshape the output of `Algorithm.fn_predict` so that it is correctly formatted for automatic post-processing. It's also more computationally efficient to do the reshaping once up front.

The `reshape_indices` argument is ultimately fed to np.reshape(newshape). We use *index n* to point to the value at `ndarray.shape[n]`.

Reshaping by Index

Let's say we have a 4D feature consisting of 3D images `(samples * channels * rows * columns)`. Our problems is that the images are B&W, so we don't want a color channel because it would add unecessary dimensionality to our model. So we want to drop the dimension at the shape index 1.

```
reshape_indices = (0,2,3)
```

Thus we have wrangled ourselves a 3D feature consisting of 2D images `(samples * rows * columns)`.

Reshaping Explicitly

But what if the dimensions we want cannot be expressed by rearranging the existing indices? If you define a number as a `str`, then that number will be used as directly as the value at that position.

So if I wanted to add an extra wrapper dimension to my data to serve as a single color channel, I would simply do:

```
reshape_indices = (0,'1',1,2)
```

> *Then couldn't I just hardcode my shapes with strings?* Yes, but `FeatureShaper` is applied to all of the splits, which are assumed to have different shapes, which is why we use the indices.

Multiplicative Reshaping

Sometimes you need to stack/nest dimensions. This requires multiplying one shape index by another.

For example, if I have a 3 separate hours worth of data and I want to treat it as 180 minutes, then I need to go from a shape of (3 hours * 60 minutes) to (180 minutes). Just provide the shape indices that you want to multiply in a `tuple` like so:

> <!> if your model is unsupervised (aka generative or self-supervised), then it must output data in *"column (aka width) last"* shape. Otherwise, automated column decoding will be applied along the wrong dimension.

4.8.1 6a. Methods

└─ `FeatureShaper.from_feature()`

```
FeatureShaper.from_feature(
    feature_id
    , reshape_indices
)
```

Argument	Type	Default	Description
feature_id	int	Required	The `Feature.id` to use
reshape_indices	tuple(int/str/tuple)	Required	See Strategies.

4.8.2 6b. Attributes

These are the fields of the FeatureShaper table

Attribute	Type	Description
reshape_indices	PickleField	See #Reshaping-by-Index.Pickle because tuple has no JSON equivalent.
column_position	IntegerField	The shape index used for columns aka width.
feature	ForeignKeyField	The Feature that reshaping is applied to.

4.9 7. Window

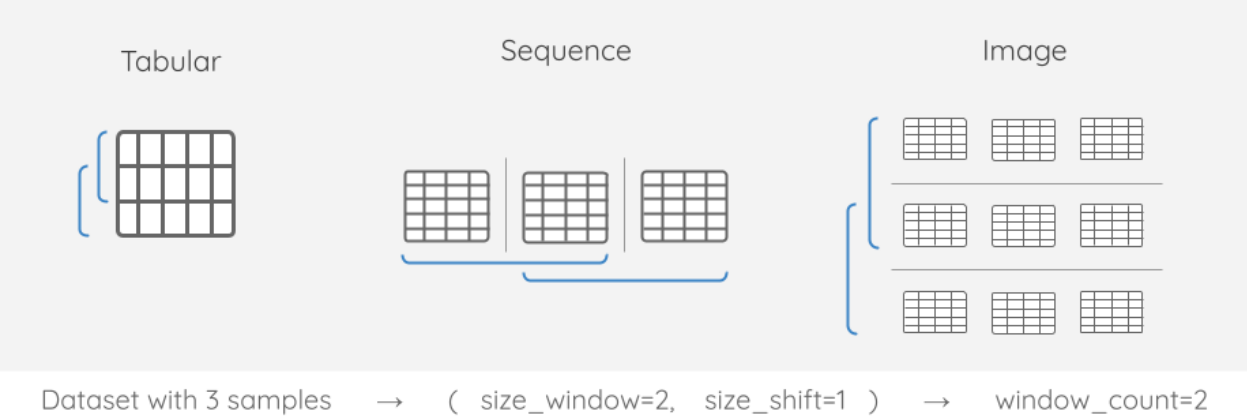
Window facilitates sliding windows for a time series Feature. It does not apply to Labels. This is used for unsupervised (aka self-supervised) walk-forward forecasting for time series data.

`size_window` determine how many timepoints are included in a window, and `size_shift` determines how many timepoints to slide over before defining a new window.

> For example, if we want to be able to *predict the next 7 days* worth of weather *using the past 21 days* of weather, then our `size_window=21` and our `size_shift=7`.

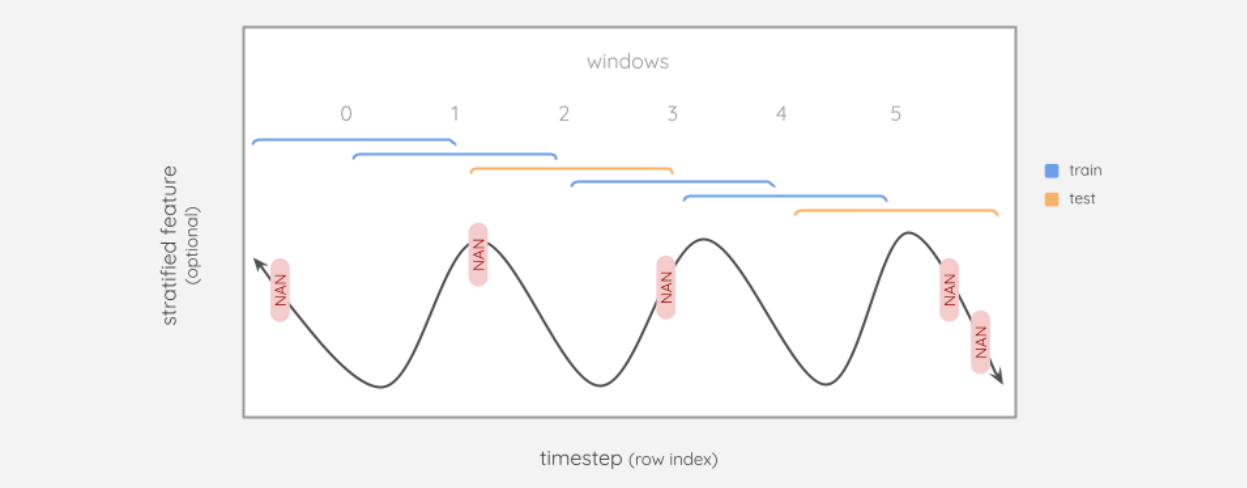
Challenges

Dealing with stratified windowed data demands a systematic approach.



Windowing always increases dimensionality

After data is windowed, its dimensionality increases by 1. Why? Well, originally we had a *single time series*. However, if we window that data, then we have *many time series subsets*.



As the highest dimension, it becomes the "sample"

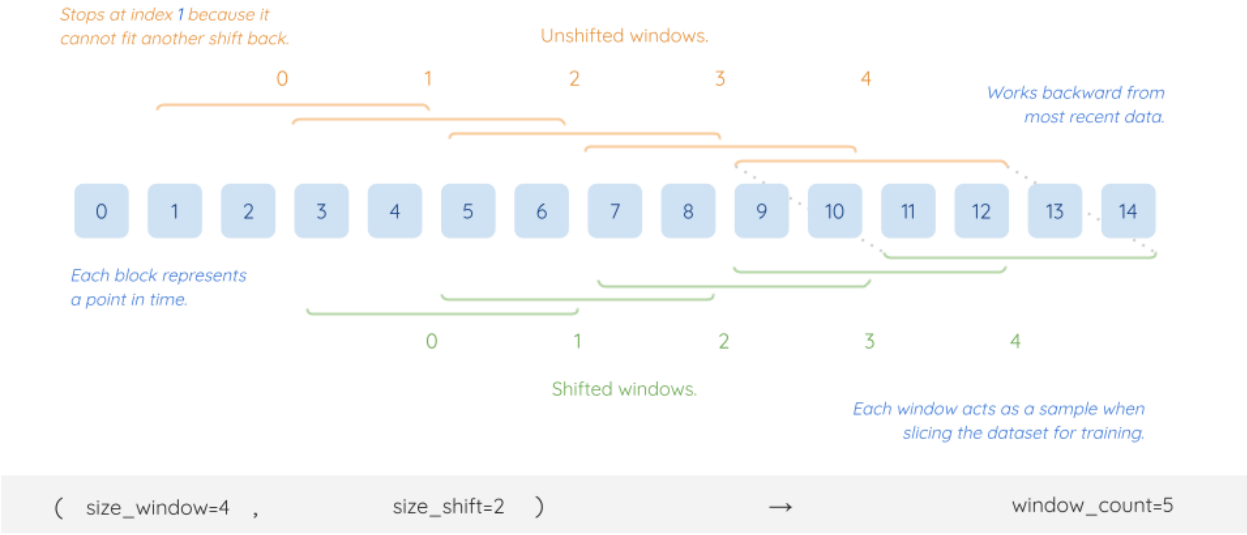
No matter what dimensionality the original data has, it will be windowed along the first dimension.

This means that the windows now serve as the samples, which is important for stratification. If we have a year's worth of windows, we don't want all of our training windows to come from the same season. Therefore, Window must be created prior to Splitset.

Windowing may causes overlap in splits

In addition to increasing the dimensionality of our data, it makes it harder to nail down the boundaries of our splits in order to prevent data leakage.

As seen in the diagram above, the timesteps of the train and test splits may *overlap*. So if we are fitting an interpolater to our training split, the first 3 NaNs would be included, but the last 2 would not.



Shifted and unshifted windows

In a walk-forward analysis, we learn about the future by looking at the past. So we need 2 sets of windows:

- *Unshifted windows* (orange in diagram above): represent the past and serves as the features we learn from
- *Shifted windows* (green in diagram above): represent the future and serves as the target we predict

However, when conducting inference, we are trying to predict the shifted windows not learn from them. So we don't need to record any shifted windows.

4.9.1 7a. Methods

└── `Window.from_feature()`

```
Window.from_feature(
    feature_id
    , size_window
    , size_shift
    , record_shifted
)
```

Argument	Type	Default	Description
dataset_id	int	Required	`Feature.id` from which you want to derive windows.
size_window	int	Required	The number of timesteps to include in a window.
size_shift	int	Required	The number of timesteps to shift forward.
record_shifted	bool	True	Whether or not we want to keep a shifted set of windows around. During pure inference, this is False.

4.9.2 7b. Attributes

These are the fields of the Window table

Attribute	Type	Description
size_window	IntegerField	Number of timesteps in each window
size_shift	IntegerField	The number of timesteps in the shift forward.
window_count	IntegerField	Not a relationship count! Number of windows in the dataset. This becomes the new samples dimension for stratification.
samples_unshifted	JSONField	Underlying sample indices of each window in the past-shifted windows.
samples_shifted	JSONField	Underlying sample indices of each window in the future-shifted windows.
feature	ForeignKeyField	The Feature that this windowing is applied to

---

4.10 8. Splitset

Used for sample stratification. Reference Stratification section of the Explainer.

Split	Description
train	The samples that the model will be trained upon. Later, we'll see how we can make *cross-folds from our training split*. Unsupervised learning will only have a training split.
validation (optional)	The samples used for training evaluation. Ensures that the test set is not revealed to the model during training.
test (optional)	The samples the model has never seen during training. Used to assess how well the model will perform on unobserved, natural data when it is applied in the real world aka how generalizable it is.

> Because Splitset groups together all of the data wrangling entities (Features, Label, Folds) it essentially represents a *Pipeline*, which is why it bears the name Pipeline in the High-Level API.

Cross-Validation

Cross-validation is triggered by `fold_count:int` during Splitset creation. Reference the scikit-learn documentation to learn more about cross-validation.

Each row in the diagram above is a `Fold` object.

Each green/blue box represents a bin of stratified samples. During preprocessing and training, we rotate which blue bin serves as the validation samples (`fold_validation`). The remaining green bins in the row serve as the training samples (`folds_train_combined`).

Let's say we defined `fold_count=5`. What are the implications?

- Creates 5 `Folds` related to a `Splitset`.
- 5x more preprocessing and caching; each `fold_validation` is excluded from the `fit` on `folds_train_combinared`. Fits are saved to the `orm.Fold` object as opposed to the `orm.Feature/Label` objects.
- 5x more models will be trained for each experiment.

- 5x more evaluation.

Disclaimer about inherent limitations & challenges

> Do not use cross-validation unless the distribution of each resulting fold (*total sample count divided by fold_count*) is representatitve of your broader sample population. If you are ignoring that advice and stretching to perform cross-validation, then at least ensure that the *total sample count is evenly divisble by fold_count*. Both of these tips help avoid poorly stratified/ undersized folds that seem to perform either unjustifiably well (100% accuracy when only the most common label class is present) or poorly (1 incorrect prediction in a small fold negatively skews an otherwise good model).
>
> If you've ever performed cross-validation manually with too few samples, then you'll know that it's easy enough to construct the folds, but then it's a pain to calculate performance metrics (e.g. `zero_division`, absent OHE classes) due to the absence of outlying classes and bins. Time has been invested to handle these scenarios elegantly so that folds can be treated as first-class-citizens alongside splits. That being said, if you try to do something undersized like multi-label classification using 150 samples then you may run into errors during evaluation.

Samples Cache

Each Splitset has as `cache_path` attribute, which represents a local directory where preprocessed data is stored during training & evaluation.

The output of `feature.preprocess()` and `label.preprocess()` are written to this folder prior to training so that:

1. Each Job does not have to preprocess data from scratch.
2. The original data does not need to be held in memory between Jobs.

```
└── aiqc/cache/samples/splitset_uid
    └── <fold_index> | "no_fold"
        └── <split>
            └── label.npy
            └── feature_<i>.npy
```

- `<fold_index>` is a folder for each Fold, since they have different samples. Whereas "no_fold" is a single folder for a regular splitset where there are no folds. 'no_fold' just keeps the folder depth uniform for regular splitsets
- `<split>`: The `samples[<split>]` of interest: 'train', 'validation', 'folds_train_combined', 'fold_validation', 'test'.
- `feature_<n>` accounts for Splitsets with more than 1 Feature.

The `Splitset.cache_hot:bool` argument indicates whether or not the cache for that splitset is populated or not.

Samples are automatically cached during `Queue.run_jobs()`.

See also: `orm.splitset.clear_cache()` and `utils.config.clear_cache_all()`

4.10.1 8a. Methods

└── `Splitset.make()`

```
Splitset.make(
    feature_ids
    , label_id
    , size_test
    , size_validation
    , bin_count
    , fold_count
    , unsupervised_stratify_col
    , name
    , description
    , predictor_id
)
```

Argument	Type	Default	Description
feature_ids	list(int)	Required	Multiple `Feature.id`'s may be included to enable multi-modal (aka mixed data-type) analysis. All of these Features must have the same number of samples.
label_id	int	None	The Label to be used as a target for supervised analysis. Must have the sample number of samples as the Features.
size_test	float	None	Percent of samples to be placed into the test split. Must be > `0.0` and < `1.0`.
size_validation	float	None	Percent of samples to be placed into the validation split. Must be > `0.0` and < `1.0`. If this is not None and used in combination with `fold_count`, then there will be 4 splits.
bin_count	int	None	For continous stratification columns, how many bins (aka quantiles) should be used?
fold_count	int	None	The number or cross-validation folds to generate. See Cross-Validation.
unsupervised_stratify_col	str	None	Used during unsupervised analysis. Specify a column from the first Feature in feature_ids to use for stratification. For example, when forecasting, it may make sense to stratify by the day of the year.
name	str	None	Used for versioning a pipeline (collection of inputs, label, and stratification). Two versions cannot have identical attributes.
description	str	None	What is unique about this this pipeline?

`size_train = 1.00 - (size_test + size_validation)` the backend ensures that the sizes sum to 1.00

How does continuous binning work? Reference the handy `Pandas.qcut()` and the source code `pd.qcut(x=array_to_bin, q=bin_count, labels=False, duplicates='drop')` for more detail.

└── `Splitset.cache_samples()`

See *Samples Cache* section for a description

```
Splitset.cache_samples(id)
```

Argument	Type	Default	Description
id	int	Required	The identifier of the Splitset of interest

└── `Splitset.clear_cache()`

See *Samples Cache* section for a description. Deletes the entire directory located at `Splitset.cache_path`.

```
Splitset.clear_cache(id)
```

Argument	Type	Default	Description
id	int	Required	The identifier of the Splitset of interest

└── `Splitset.fetch_cache()`

See *Samples Cache* section for a description. This fetches a specific file from the cache.

```
Splitset.fetch_cache(
    id
    , split
    , label_features
    , fold_id
    , library
)
```

Argument	Type	Default	Description
id	int	Required	The identifier of the Splitset of interest
split	int	Required	The `samples[<split>]` of interest: 'train', 'validation', 'folds_train_combined', 'fold_validation', 'test'.
label_features	str	Required	Either `'label'` or `'features'`
fold_id	int	None	The identifier of the Fold of interest, if any
library	str	None	If `'pytorch'`, it will convert each returned array to `FloatTensor()`

`label_features` Value	Returns
`'label'`	`ndarray`
`'features'`	`ndarray` or `list(ndarray)`

4.10.2 8b. Attributes

These are the fields of the Splitset table

Attribute	Type	Description
cache_path	CharField	Where the splitset stores its cached samples
cache_hot	BooleanField	If the samples are currently stored in the cache
samples	JSONField	The bins that splits have been stratified into `dict(split=[sample_indice s])`
sizes	JSONField	Human-readable sizes of the splits `dict(split=dict(percent=f loat, count=int))`
supervision	CharField	Either "supervised" or "unsupervised" if the Splitset has a Label.
has_validation	BooleanField	Logical flag indicating if this Splitset has a validation split.
fold_count	IntegerField	The number of cross-validation `Folds` that belong to this Splitset
bin_count	IntegerField	The number of bins used to stratify a continuous column label or unsupervised_stratify column
**unsupervised_stratifyCol **	CharField	Used during unsupervised analysis. Specify a column from the first Feature in feature_ids to use for stratification. For example, when forecasting, it may make sense to stratify by the day of the year.
key_train	CharField	`'train'` by default, but `'folds_train_combined'` if Splitset has Folds. `None` for an inference splitset.
key_evaluation	CharField	`'test'` if neither validation split nor Folds are used. `'validation'` if a validation split is used. `'fold_validation'` if Splitset has Folds. `None` for an inference splitset.
key_test	CharField	`'test'` by default. `None` for an inference splitset.
version	IntegerField	[TBD]
label	ForeignKeyField	The Label, if any, that supervises this splitset
predictor	DeferredForeignKey	During inference, a new Splitset of samples to be predicted may attach to a Predictor. Samples dict will bear the key of the `Predictor.Splitset.count( )` e.g. `'infer_0'`.

These are the fields of the Fold table

Attribute	Type	Description
idx	IntegerField	Zero-based auto-incrementer that counts the Folds
samples	JSONField	Contains the sample indices of the training folds and leftout validation fold, as well as any validation and test splits defined in the regular Splitset.samples `dict().keys()==['folds_train_combined', 'fold_valida tion', 'validation', 'test' ]`
fitted_labelcoder	PickleField	When `LabelCoders`'s `fit` an sklearn preprocessor, the fit objects are saved here for downstream `inverse_transform`'ing
fitted_featurecoders	PickleField	When `FeatureCoder`'s `fit` an sklearn preprocessor, the fit objects are saved here for downstream `inverse_transform`'ing
splitset	ForeignKeyField	The Splitset that this Fold belongs to

4.11 9. Algorithm

Now that our data has been prepared, we transition to the 2nd half of the ORM where the focus is the logic that will be applied to that data.

The Algorithm contains all of the components needed to construct, train, and use our model.

Reference the tutorials for examples of how Algorithms are defined.

PyTorch Fit

Provides an abstraction that eliminates the boilerplate code normally required to train and evaluate a PyTorch model.

- Before training - it shuffles samples, batches samples, and then shuffles batches.
- During training - it calculates batch loss, epoch loss, and epoch history metrics.
- After training - it calculates metrics for each split.

```
model, history = utils.pytorch.fit(
    # These arguments come directly from `fn_train`
    model
    , loser
    , optimizer

    , train_features
    , train_label
    , eval_features
    , eval_label

    # These arguments are user-defined
    , epochs
    , batch_size
    , enforce_sameSize
    , allow_singleSample
    , metrics
)
```

User-Defined Arguments	Type	Default	Description
epochs	int	30	The number of times to loop over the features
batch_size	int	5	Divides features and lables into chunks to be trained upon
enforce_sameSize	bool	True	If `True`, drops `len(batch!=batch_size)`
allow_singleSample	bool	False	If `False`, drops `len(batch!=1)`
metrics	list(torchmetrics.metric())	None	List of instantiated `torchmetrics` classes e.g. `Accuracy`

History Metrics

The goal of the `Predictor.history` object is to record the training and evaluation metrics at the end of each epic so that they can be interpretted in the learning curve plots. Reference the evaluation section.

- *Keras*: any `metrics=[]` specified are automatically added to the `History` callback object.

- *PyTorch*: if you use `fit` seen above, then you don't need to worry about this. Users are responsible for calculating their own metrics (we recommend the `torchmetrics` package) and placing them into a `history` dictionary that mirrors the schema of the Keras history object. Reference the torch examples.

  The schema of the `history` dictionary is as follows: `dict(<metric>:ndarray, val_<metric>=ndarray)`. For example, if you wanted to record the history of the 'loss' and 'accuracy' metrics manually for PyTorch, you would construct it like so:

```
history = dict(
    loss           = ndarray
    , val_loss     = ndarray

    , accuracy     = ndarray
    , val_accuracy = ndarray
)
```

TensorFlow Early Stopping

Early stopping isn't just about efficiency in reducing the number of `epochs`. If you've specified 300 epochs, there's a chance your model catches on to the underlying patterns early, say around 75-125 epochs. At this point, there's also good chance what it learns in the remaining epochs will cause it to overfit on patterns that are specific to the training data, and thereby and lose it's simplicity/ generalizability.

> The `metric=val_*` prefix refers to the evaluation samples.
>
> Remember, regression does not have accuracy metrics.
>
> `TrainingCallback.MetricCutoff` is a custom class we wrote to make *early stopping* easier, so you won't find information about it in the official Keras documentation.

Placed within `fn_train`:

```
from aiqc.utils.tensorflow import TrainingCallback

#Define one or more metrics to monitor.
metrics_cuttoffs = [
    dict(metric='accuracy', cutoff=0.96, above_or_below='above'),
    dict(metric='loss', cutoff=0.1, above_or_below='below')
    dict(metric='val_accuracy', cutoff=0.96, above_or_below='above'),
    dict(metric='val_loss', cutoff=0.1, above_or_below='below')
]
cutoffs = TrainingCallback.MetricCutoff(metrics_cuttoffs)

# Pass it into keras callbacks
model.fit(
    # other fit args
    callbacks = [cutoffs]
)
```

> Tip: try using a `val_accuracy` threshold by itself for best results

4.11.1 9a. Methods

Assemble an architecture consisting of components defined in functions.

The `**hp` kwargs are common to every Algorithm function except `fn_predict`. They are used to systematically pass a dictionary of *hyperparameters* into these functions. See Hyperparameters.

└── `Algorithm.make()`

```
Algorithm.make(
    library
    , analysis_type
    , fn_build
    , fn_train
    , fn_predict
    , fn_lose
    , fn_optimize
)
```

Argument	Type	Default	Description
library	str	Required	'keras' or 'pytorch' depending on the type of model defined in `fn_build`
analysis_type	str	Required	'classification_binary', 'classification_multi', or 'regression'. Unsupervised/ self-supervised falls under regression. Used to determine which performance metrics are run. Errors if it is incompatible with the Label provided: e.g. classification_binary is incompatible with an np.floating Label.column.
fn_build	func	Required	See below. Build the model architecture.
fn_train	func	Required	See below. Train the model.
fn_predict	func	None	See below. Run the model.
fn_lose	func	None	See below. Calculate loss.
fn_optimize	func	None	See below. Optimization strategy.

Required Functions

```
def fn_build(
    features_shape:tuple
    , label_shape:tuple
    , **hp:dict
):
    # Define tf/torch model
    return model
```

> The `*_shape` arguments contain the shape of a single sample, as opposed to a batch or entire dataset. `features_shape` is plural because it may contain the shape of multiple features. However, if only 1 feature was used then it will not be inside a list.

```
def fn_train(
    model:object
```

(continues on next page)

(continued from previous page)

```
    , loser:object
    , optimizer:object
    , train_features:ndarray
    , train_label:ndarray
    , eval_features:ndarray
    , eval_label:ndarray
    , **hp:dict
):
    # Define training/ eval loop.
    # See `utils.pytorch.fit`

    # if tensorflow
    return model
    # if torch
    # See `utils.pytorch.fit` and history metrics below
    return history:dict, model
```

Optional Functions

Where are the defaults for optional functions defined? See utils.tensorflow and utils.pytorch for examples of loss, optimization, and prediction.

```
def fn_predict(model:object, features:ndarray):
    #if classify. predictions always ordinal, never OHE.
    return prediction, probabilities #both as ndarray

    #if regression
    return prediction #ndarray
```

```
def fn_lose(**hp:dict):
    # Define tf/torch loss function
    return loser
```

```
def fn_optimize(**hp:dict):
    # Define tf/torch optimizer
    return optimizer
```

└── `Algorithm.get_code()`

Returns the strings of the Algorithm functions:

```
dict(
    fn_build      = aiqc.utils.dill.reveal_code(Algorithm.fn_build)
    , fn_lose     = aiqc.utils.dill.reveal_code(Algorithm.fn_lose)
    , fn_optimize = aiqc.utils.dill.reveal_code(Algorithm.fn_optimize)
    , fn_train    = aiqc.utils.dill.reveal_code(Algorithm.fn_train)
    , fn_predict  = aiqc.utils.dill.reveal_code(Algorithm.fn_predict)
)
```

4.11.2 9b. Attributes

These are the fields of the Algorithm table

Attribute	Type
library	CharField
analysis_type	CharField
fn_build	BlobField
fn_lose	BlobField
fn_optimize	BlobField
fn_train	BlobField
fn_predict	BlobField

See #9.-Algorithm for descriptions

4.12 10. Hyperparameters

As mentioned in Algorithm, the `**hp` argument is used to systematically pass hyperparameters into the Algorithm functions.

For example, given the follow set of hyperparamets:

```
hyperparameters = dict(
    epoch_count     = [30]
    , learning_rate = [0.01]
    , neuron_count  = [24, 48]
)
```

A grid search would produce the 2 unique `Hyperparamcombo`'s:

```
[
    dict(
        epoch_count     = 30
        , learning_rate = 0.01
        , neuron_count  = 24 #<-- varies
    )

    , dict(
        epoch_count     = 30
        , learning_rate = 0.01
        , neuron_count  = 48 #<-- varies
    )
]
```

We access the current value in our model functions like so: `hp['neuron_count']`.

4.12.1 10a. Methods

└── `Hyperparamset.from_algorithm()`

```
Hyperparamset.from_algorithm(
    algorithm_id
    , hyperparameters
    , search_count
    , search_percent
)
```

Argument	Type	Default	Description
algorithm_id	int	Required	The `Algorithm.id` whose functions these hyperparameters will be used with
hyperparameters	dict(str:list)	Required	See example in Hyperparameters. Must be JSON compatible.
search_count	int	None	Randomly select *n* hyperparameter combinations to test. Must be greater than 1. No upper limit, it will test all combinations if number of combinations is exceeded.
search_percent	float	None	Given all of the available hyperparameter combinations, search x%. Between `0.0: 1.0`. Cannot be used if `search_count` is used.

> "Bayesian TPE (Tree-structured Parzen Estimator)" via `hyperopt` has been suggested as a future area to explore, but it does not exist right now.

4.12.2 10b. Attributes

These are the fields of the Hyperparamset table

Attribute	Type	Description
hyperparameters	JSONField	The original `dict(param=list)` of all possible values
search_count	IntegerField	The number of randomly selected combinations of hyperparameters
search_percent	FloatField	The percent of randomly selected combinations of hyperparameters
algorithm	ForeignKeyField	The `Algorithm.id` whose functions these hyperparameters will be used with

These are the fields of the Hyperparamcombo table

Attribute	Type	Description
idx	IntegerField	Zero-based counts the number of the number of hyperparamcombos
hyperparameters	JSONField	The specific combination of hyperparameters that will be fed to the Algorithm functions
hyperparamset	ForeignKeyField	The Hyperparamset that this combination of hyperparameters was derived from

---

4.13 11. Queue

The Queue is the central object of the "logic side" of the ORM. It ties together everything we need to run training `Job`'s for hyperparameter tuning. That's why it is referred to as an Experiment in the High-Level API.

4.13.1 11a. Methods

└── `Queue.from_algorithm()`

```
Queue.from_algorithm(
    algorithm_id
    , splitset_id
    , repeat_count
    , permute_count
    , hyperparamset_id
    , description
)
```

Argument	Type	Default	Description
algorithm_id	int	Required	The `Algorithm.id` whose functions will be used during training and evaluation
splitset_id	int	Required	The `Splitset.id` whose samples will be used during training and evaluation
repeat_count	int	1	Each job will be repeat n times. Designed for use with random weight initialization (aka non-deterministic). This is why 1 `Job` has many `Predictors`
permute_count	int	3	Triggers a shuffled permutation of each training data column to determine which columns have the most impact on loss in comparison baseline training loss: `[training loss - (median loss of <n> permutations)]` `. The count determines how many times the shuffled permutation is ran before taking the median loss. Permutation does *not* get run on ``Feature.dataset.typ=='ima ge'`. Set this to 0 if you do not care about feature importance. Retroactive feature importance is possible via `Prediction.calcFeatureImp ortance()`.
hyperparamset_id	int	None	The `Hyperparamset.id` whose samples will be used during training and evaluation. This needs to be specified because an Algorithm can have many Hyperparamsets.
description	str	None	What is unique about this experiment?

└── `Queue.run_jobs()`

Jobs are simply ran on a loop on the main process.

Stop the queue with a keyboard interrupt e.g. `ctrl+Z/D/C` in Python shell or `i,i` in Jupyter. It is listening for interupts so it will usually stop gracefully. Even if it errors upon during interrupt, it's not a problem. You can rerun the queue and it will resume on the same job it was running previously.

```
Queue.run_jobs(id)
```

Argument	Type	Default	Description
id	int	Required	The identifier of the Queue of interest

└── `Queue.plot_performance()`

Plots every model trained by the queue for comparison.

- X axis = loss
- Y axis = score

```
Queue.plot_performance(
    id
    , call_display
    , max_loss
    , min_score
    , score_type
    , height
)
```

Argument	Type	Default	Description
id	int	Required	The identifier of the Queue of interest
call_display	bool	True	If `True`, calls `display()` on plot. If `False`, returns Plotly `figure` object.
max_loss	float	None	Models with any split with higher loss than this threshold will not be plotted.
min_score	typ	None	Models with any split with a lower score than this threshold will not be plotted.
score_type	typ	None	Defaults to `"accuracy"` for classification analysis, and `"r2"` for regression analysis. See aiqc.utils.meter for available metrics.
height	typ	None	Default height is `560` but you can force it to be taller

---

└── `Queue.metrics_df()`

Displays metrics for every split/fold of every model.

```
Queue.metrics_df(
    id
    , selected_metrics
    , sort_by
    , ascending
)
```

Argument	Type	Default	Description
id	int	Required	The identifier of the Queue of interest
ascending	typ	False	Descending if False.
selected_metrics	list(str)	None	If you get overwhelmed by the variety of metrics returned, then you can include the ones you want selectively by name.
sort_by	str	None	You can sort the dataframe by any column name.

└── `Queue.metricsAggregate_df()`

Aggregate statistics about every metric of every model trained in the Queue – displays the average, median, standard deviation, minimum, and maximum across all splits/folds.

```
Queue.metricsAggregate_df(
    id
    , ascending        = False
    , selected_metrics = None
    , selected_stats   = None
    , sort_by          = None
)
```

Argument	Type	Default	Description
id	int	Required	The identifier of the Queue of interest
ascending	typ	False	Descending if False.
selected_metrics	list(str)	None	If you get overwhelmed by the variety of metrics returned, then you can include the ones you want selectively by name.
sort_by	str	None	You can sort the dataframe by any column name.

4.13.2 11b. Attributes

These are the fields of the Queue table

Attribute	Type	Description
repeat_count	IntegerField	The number of times to repeat each Job.
total_runs	IntegerField	The total number of models to be trained as a result of this queue being created.
permute_count	IntegerField	Number of permutations to run on each column before taking the median impact on loss. 0 means permutation was skipped.
runs_completed	IntegerField	Counts the runs that have actually finished
algorithm	ForeignKeyField	The model functions to use during training and evaluation
splitset	ForeignKeyField	The pipeline of samples to feed to the models during training and evaluation
hyperparamset	ForeignKeyField	Contains all of the hyperparameters to be used for the Jobs

---

4.14 12. Job

The Queue spawns `Job`'s. A Job is like a spec/ manifest for training a model. It may be repeated.

```
# jobs = Hyperamset.hyperamcombo.count() * Queue.repeat_count * splitset.folds.count()
```

4.14.1 12a. Methods

There are no noteworthy, user-facing methods for the Job class

---

4.14.2 12b. Attributes

These are the fields of the Job table

Attribute	Type	Description
repeat_count	IntegerField	The number of times this Job is to be repeated
queue	ForeignKeyField	The Queue this Job was created by
hyperparamcombo	ForeignKeyField	The parameters this Job uses
fold	ForeignKeyField	The cross-validation samples that this Job uses

---

4.15 13. Predictor

As the Jobs finish, they save the `model` and `history` metrics within a `Predictor` object.

4.15.1 13a. Methods

└── `Predictor.get_model()`

```
predictor.get_model(id)
```

Handles fetching and initializing the model (and PyTorch optimizer) from `Predictor.model_file` and `Predictor.input_shapes`

Argument	Type	Default	Description
id	int	None	The identifier of the Predictor of interest

└── `Predictor.get_hyperparameters()`

This is a shortcut to fetch the hyperparameters used to train this specific model. `as_pandas` toggles between `dict()` and `DataFrame`.

```
Predictor.get_hyperparameters(id, as_pandas)
```

Argument	Type	Default	Description
id	int	None	The identifier of the Predictor of interest
as_pandas	bool	True	If `True`, returns a DataFrame. If `False`, returns a `list(dict(param_name=[values]))`

└── `Predictor.plot_learning_curve()`

A learning curve will be generated for each train-evaluation pair of metrics in the Predictor.history dictionary

```
Predictor.plot_learning_curve(
    id
    , skip_head
    , call_display
)
```

Argument	Type	Default	Description
id	int	None	The identifier of the Predictor of interest
skip_head	bool	True	Skips displaying the first 15% of epochs. Loss values in the first few epochs can often be extremely high before they plummet and become more gradual. This really stretches out the graph and makes it hard to see if the evaluation set is diverging or not.
call_display	bool	True	If `True`, calls `display()` on plot. If `False`, returns Plotly `figure` object(s).

4.15.2 13b. Attributes

These are the fields of the Predictor table

Attribute	Type	Description
repeat_index	IntegerField	Counts how many predictors have been trained using a Job spec
time_started	DateTimeField	When the Job started
time_succeeded	DateTimeField	When the Job finished
time_duration	IntegerField	Total time in seconds it took to complete the Job
model_file	BlobField	Contains a dilled (advanced Pickle) of the trained model. See `Predictor.get_model()` for exporting.
features_shapes	PickleField	tuple or list of tuples containing the np.shape(s) of feature(s)
label_shape	PickleField	tuple containing np.shape of a single sample's label
history	JSONField	Contains the training history loss/metrics
is_starred	BooleanField	Flag denoting if this model is of interest
job	ForeignKeyField	The Job that trained this Predictor

4.16 14. Prediction

When data is fed through a Predictor, you get a `Prediction`. During training, Predictions are automatically generated for every split/fold in the `Queue.splitset`.

4.16.1 14a. Methods

└── `Prediction.calc_featureImportance()`

This method is provided for conducting feature importance after training. It was decoupled from training for the following reasons:

- Permutation is computationally expensive, especially for many-columned datasets.
- We don't care about the feature importance of our best models.

What data is used when calculating feature importance? All splits/folds are concatenated back into a single dataset. This assumes that all splits/folds are relatively equally balanced with respect to their label values. For example, if you have unbalanced multi-labels (55:35:10 distribution of classes) then a given feature's importance may be biased based on how well it predicts the larger class. For binary classification scenarios, this should not matter as much since predicting one class also helps in predicting the opposite class.

Upon completion it will update the `Prediction.feature_importance` and `Prediction.permute_count` attributes.

```
Prediction.calc_featureImportance(id, permute_count)
```

Argument	Type	Default	Description
id	int	None	The identifier of the Prediction of interest
permute_count	int	Required	The count determines how many times the shuffled permutation is ran before taking the median loss. [training loss - (median loss of <n> permutations)]` ` Permutation *skips* ``Feature.dataset.typ=='image'.

└─ Prediction.importance_df()

Returns a dataframe of feature columns ranked by their median importance

```
Prediction.importance_df(id, top_n, feature_id)
```

Argument	Type	Default	Description
id	int	None	The identifier of the Prediction of interest
top_n	int	None	The number of columns to return
feature_id	int	None	Limit returned columns to a specific feature identifier

---

└─ Prediction.plot_feature_importance()

Plots prediction.feature_importance if Queue.permute_count>0 or Prediction.calc_featureImportance() was ran after the fact.

```
Prediction.plot_feature_importance(
    id
    , call_display
    , top_n
    , height
    , margin_left
)
```

Argument	Type	Default	Description
id	int	None	The identifier of the Prediction of interest
call_display	bool	True	If True, calls display() on plot. If False, returns Plotly figure object.
top_n	int	10	The number of features to display. If greater than the actual number of features, it just returns all features.
box-points	object	False	Determines how whiskers, outliers, and points are shown. Options are: False, 'all', 'suspectedoutliers', and 'outliers'. Reference Plotly Box Plots.
height	int	None	If None, dynamically makes the plot taller to fit all of the columns
margin_left	int	None	If None, dynamically makes the y axis margin wider the longest column name

---

└─ Prediction.plot_roc_curve()

Receiver operating curve (ROC) for classification metrics.

```
Prediction.plot_roc_curve(id, call_display)
```

Argument	Type	Default	Description
id	int	None	The identifier of the Prediction of interest
call_display	bool	True	If `True`, calls `display()` on plot. If `False`, returns Plotly `figure` object.

└── `Prediction.plot_precision_recall()`

Precision/recall curve for classification metrics.

```
Prediction.plot_precision_recall(id, call_display)
```

Argument	Type	Default	Description
id	int	None	The identifier of the Prediction of interest
call_display	bool	True	If `True`, calls `display()` on plot. If `False`, returns Plotly `figure` object.

└── `Prediction.plot_confusion_matrix()`

Confusion matrices for classification metrics.

```
Prediction.plot_confusion_matrix(id, call_display)
```

Argument	Type	Default	Description
id	int	None	The identifier of the Prediction of interest
call_display	bool	True	If `True`, calls `display()` on plot. If `False`, returns Plotly `figure` object(s).

└── `Prediction.plot_confidence()`

Plot the binary/multi-label classification probabilities for a single sample.

```
Prediction.plot_confidence(
    id,
    , prediction_index
    , height
    , call_display
    , split_name
)
```

Argument	Type	Default	Description
id	int	None	The identifier of the Prediction of interest
prediction_index	int	0	The index of the sample of interest
height	int	175	Force the height of the chart.
call_display	bool	True	If `True`, returns a DataFrame. If `False`, returns a `list(dict(param_name=[val ues]))`
split_name	int	None	The identifier of the Prediction of interest

---

4.16.2 14b. Attributes

Attribute	Type	Description
predictions	PickleField	Decoded predictions ndarray for per split/ fold/ inference
permute_count	IntegerField	The number of times this feature importance permuted each column
feature_importance	JSONField	Importance of each column. Only calculated for training split/fold. Schema: `dict(str(feature.id)=dict (median=float,loss_impacts= list(float)))`
probabilities	PickleField	Prediction probabilities per split/ fold. `None` for regression. Schema: `dict(split=ndarray)`
metrics	PickleField	Statistics for each split/fold that vary based on the analysis_type.
metrics_aggregate	PickleField	Contains the average, median, standard deviation, minimum, and maximum for each statistic across all splits/folds.
plot_data	PickleField	Metrics reformatted for plot functions.

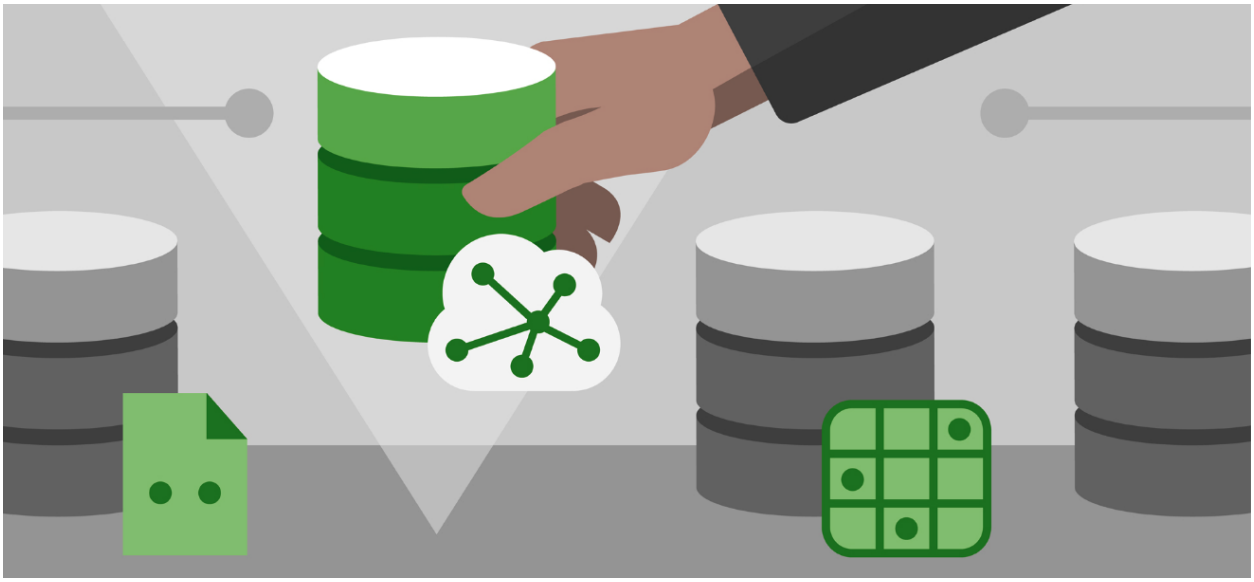
4.17 Evaluation

To see the visualization of performance metrics of `Queue`, `Predictor` and `Prediction` in action – reference the Evaluation documentation.

CHAPTER

FIVE

DATASETS



5.1 Overview

This notebook contains information about the prepackaged datasets that are referenced throughout the documentation. These datasets are either:

- Included directly within the AIQC Python package itself (KB)
- Stored remotely in the AIQC GitHub repository (MB)

5.2 Prerequisites

If you've already completed the instructions on the Installation page, then let's get started.

[2]:
```
from aiqc import datum
```

The module for interacting with the datasets is called `datum` so that it does not overlap with commonly used names like 'data' or 'datasets'.

5.3 Prepackaged Local Data

The `list_datums()` method provides metadata about each of file that is included in the package, so that you can find one that suits your purposes.

By default it returns a Pandas DataFrame, but you can `list_datums(format='list')` to change that.

[3]:
```
datum.list_datums()
```

[3]:
```
                    name dataset_type          analysis_type         label  \
0      exoplanets.parquet      tabular             regression  SurfaceTempK
1   heart_failure.parquet      tabular             regression          died
2                iris.tsv      tabular   classification_multi       species
3               sonar.csv      tabular  classification_binary        object
4              houses.csv      tabular             regression         price
5      iris_noHeaders.csv      tabular   classification multi       species
6            iris_10x.tsv      tabular   classification multi       species
7         brain_tumor.csv        image  classification_binary        status
8   galaxy_morphology.tsv        image  classification_binary    morphology
9                spam.csv         text  classification_binary            v1
10       epilepsy.parquet     sequence  classification_binary       seizure
11  delhi_climate.parquet     sequence            forecasting           N/A
12       liberty_moon.csv        image            forecasting           N/A

   label_classes                       features  samples  \
0            N/A                              8      433
1              2                             12      299
2              3                              4      150
3              2                             60      208
4            N/A                             12      506
5              3                              4      150
6              3                              4     1500
7              2   1 color x 160 tall x 120 wide       80
8              2  3 colors x 240 tall x 300 wide       40
9              2                      text data     5572
10             2             1 x 178 readings     1000
11           N/A                              3     1575
12           N/A     1 color x 50 tall x 60 wide       15


↪                                                                                ␣
↪                                                                                ␣
↪                                        description  \
0                                                                                ␣
↪                                                                                ␣
↪                     Predict temperature of exoplanet.
1                                                                                ␣
↪                                                                                ␣
↪                    Biometrics to predict loss of life.
2                                                                                ␣
↪                                                                  3 species of flowers.␣
↪Only 150 rows, so cross-folds not represent population.
3                                                                                ␣
↪                                                                    Detecting either a␣
↪rock "R" or mine "M". Each feature is a sensor reading.
```

(continues on next page)

(continued from previous page)

```
4                                                                                    ␣
↪                                                                                    ␣
↪                         Predict the price of the house.
5                                                                                    ␣
↪                                                                                    ␣
↪                           For testing; no column names.
6                                                                                    ␣
↪                                                                               For␣
↪testing; duplicated 10x so cross-folds represent population.
7                                                                                    ␣
↪                                                                                    ␣
↪           csv acts as label and manifest of image urls.
8                                                                                    ␣
↪                                                                                    ␣
↪           tsv acts as label and manifest of image urls.
9                                                                                    ␣
↪                                                                                    ␣
↪             collection of spam/ ham (not spam) messages
10  <https://archive.ics.uci.edu/ml/datasets/Epileptic+Seizure+Recognition> Storing the␣
↪data tall so that it compresses better.`label_df = df[['seizure']]; sensor_arr3D = df.
↪drop(columns=['seizure']).to_numpy().reshape(1000,178,1)`
11                <https://www.kaggle.com/sumanthvrao/daily-climate-time-series-data>.␣
↪Both train and test (pruned last day from train). 'pressure' and 'wind' columns seem␣
↪to have outliers. Converted 'date' column to 'day_of_year.'
12                                                                                   ␣
↪                                                                                    ␣
↪               moon glides from top left to bottom right

   location
0     local
1     local
2     local
3     local
4     local
5     local
6     local
7    remote
8    remote
9     local
10    local
11    local
12   remote
```

The location where Python packages are installed varies from system to system so `pkg_resources` is used to find the location of these files dynamically.

Using the value of the name column, you can fetch that file via `to_pandas()`.

[4]:
```
df = datum.to_pandas(name='houses.csv')
df.head()
```

[4]:
```
      crim    zn  indus  chas    nox     rm   age     dis  rad  tax  ptratio  \
0  0.00632  18.0   2.31     0  0.538  6.575  65.2  4.0900    1  296     15.3
```

(continues on next page)

(continued from previous page)

```
1  0.02731   0.0   7.07     0  0.469  6.421  78.9  4.9671    2  242     17.8
2  0.02729   0.0   7.07     0  0.469  7.185  61.1  4.9671    2  242     17.8
3  0.03237   0.0   2.18     0  0.458  6.998  45.8  6.0622    3  222     18.7
4  0.06905   0.0   2.18     0  0.458  7.147  54.2  6.0622    3  222     18.7

   lstat  price
0   4.98   24.0
1   9.14   21.6
2   4.03   34.7
3   2.94   33.4
4   5.33   36.2
```

Alternatively, if you prefer to work directly with the file itself, then you can obtain the location of the file via the `get_demo_file_path()` method.

[5]:
```
datum.get_path('houses.csv')
```

[5]:
```
'/Users/layne/Desktop/AIQC/aiqc/data/houses.csv'
```

5.4 Remote Data

In order to avoid bloating the package, larger dummy datasets are *not* included in the package. These kind of datasets consist of many large files.

They are located within the repository at `https://github.com/aiqc/aiqc/remote_datum`.

> If you want to fetch them on your own, you can do so by appending `?raw=true` to the end of an individual file's URL.
>
> Otherwise, their locations are hardcoded into the datum methods.

We'll use the `brain_tumor.csv` file as an example.

[6]:
```
df = datum.to_pandas(name='brain_tumor.csv')
df.head()
```

[6]:
```
   status  size  count symmetry  \
0       0     0      0      NaN
1       0     0      0      NaN
2       0     0      0      NaN
3       0     0      0      NaN
4       0     0      0      NaN


↪           url
0  https://raw.githubusercontent.com/aiqc/aiqc/main/remote_datum/image/brain_tumor/
↪images/healthy_0.jpg
1  https://raw.githubusercontent.com/aiqc/aiqc/main/remote_datum/image/brain_tumor/
↪images/healthy_1.jpg
2  https://raw.githubusercontent.com/aiqc/aiqc/main/remote_datum/image/brain_tumor/
↪images/healthy_2.jpg
```

(continues on next page)

(continued from previous page)

```
3  https://raw.githubusercontent.com/aiqc/aiqc/main/remote_datum/image/brain_tumor/
↪images/healthy_3.jpg
4  https://raw.githubusercontent.com/aiqc/aiqc/main/remote_datum/image/brain_tumor/
↪images/healthy_4.jpg
```

This file acts as a manifest for multi-modal datasets in that each row of this CSV represents a sample:

- The `'status'` column of this file serves as the Label of that sample. We'll construct a `Dataset.Tabular` from this.
- Meanwhile, the `'url'` column acts as a manifest in that it contains the URL of the image file for that sample. We'll construct a `Dataset.Image` from this.

[7]:
```
img_urls = datum.get_remote_urls('brain_tumor.csv')
img_urls[:3]
```

[7]:
```
['https://raw.githubusercontent.com/aiqc/aiqc/main/remote_datum/image/brain_tumor/images/
↪healthy_0.jpg',
 'https://raw.githubusercontent.com/aiqc/aiqc/main/remote_datum/image/brain_tumor/images/
↪healthy_1.jpg',
 'https://raw.githubusercontent.com/aiqc/aiqc/main/remote_datum/image/brain_tumor/images/
↪healthy_2.jpg']
```

At this point, you can use either the high or low level AIQC APIs [e.g. `aiqc.Dataset.Image.from_ulrs()`] to ingest that data and work with it as normal.

5.5 Alternative Sources

If the datasets described above do not satisfy your use case, then have a look at the following repositories:

- UCI: https://archive.ics.uci.edu/ml/datasets.php
- Kaggle: https://www.kaggle.com/datasets
- Quilt: https://open.quiltdata.com/
- sklearn: https://scikit-learn.org/stable/datasets/toy_dataset.html
- seaborn: https://github.com/mwaskom/seaborn-data

CHAPTER

SIX

EVALUATION

6.1 Overview

Every training `Job` automatically generates metrics when evaluated against each split/ fold.

All Analyses

Loss is every neural network's measure of overall prediction error. The lower the loss, the better. However, it's not really intuitive for humans, which is why analysis specific metrics like accuracy and R^2 are necessary.

Metrics	loss
Plots	boomerang plot, learning curve, feature importance

Classification

Although `'classification_multi'` and `'classification_binary'` share the same metrics and plots, they go about producing these artifacts differently: e.g. ROC curves `roc_multi_class=None` vs `roc_multi_class='ovr'`.

Metrics	accuracy, f1, roc_auc, precision, recall, probabilities
Plots	ROC-AUC, precision-recall, confusion matrix, sigmoid/ pie probabilities

Regression

Does not have an `'accuracy'` metric, so we default to `'r2'`, R^2 (coefficient of determination, as a guage of effectiveness. There are no regression-specific plots in AIQC yet. Note that, as a quantitative measure of similarity, unsupervised/ self-supervised models are also considered a regression.

Metrics	r2, mse, explained_variance

Dashboard Arguments

In order to accomodate the dashboards, the following arguments were added:

- `call_display:bool=True` when `True`, performs `figure.display()`. Whereas when `False`, it returns the raw Plotly `figure` object. The learning curve, feature importance, and confusion matrix functions return `list(figs)`.
- `height:int=None` pixel-based adjustment for boomerang chart and feature importance.

  The actual arguments of the methods in this in this notebook are documented in the Low-Level Docs,

---

6.2 Prerequisites

`Plotly` is used for interactive charts (hover, toggle, zoom). Reference the Installation section for information about configuring Plotly. However, static images are used in this notebook due to lack of support for 3rd party JS in the documentation portal.

We'll use the `datum` and `tests` modules to rapidly generate a couple examples.

[2]:
```
from aiqc import datum
from aiqc import tests
```

6.3 Classification

Let's quickly generate a trained classification model to inspect.

[3]:
```
%%capture
queue_multiclass = tests.tf_multi_tab.make_queue()
queue_multiclass.run_jobs()
```

6.3.1 Queue Visualization

`plot_performance` aka the boomerang chart is unique to AIQC, and it really brings the benefits of the library to light. Each model from the Queue is evaluated against all splits/ folds.

When evaluating a classification-based `Queue.analysis_type`, the following `score_type:str` are available: accuracy, f1, roc_auc, precision, and recall.

[ ]:
```
queue_multiclass.plot_performance(
    max_loss = 1.5, score_type='accuracy', min_score = 0.70
)
```

6.3.2 Queue Metrics

[5]:
```
queue_multiclass.metrics_df(
    selected_metrics = None
    , sort_by        = 'predictor_id'
    , ascending      = True
).head(6)
```

[5]:
```
   hyperparamcombo_id  job_id  predictor_id       split  accuracy     f1  \
0                  17      23            23       train     0.912  0.911
1                  17      23            23  validation     0.810  0.806
2                  17      23            23        test     0.963  0.963

    loss  precision  recall  roc_auc
0  0.271      0.917   0.912    0.983
1  0.317      0.822   0.810    0.966
2  0.240      0.967   0.963    1.000
```

These are also aggregated by metric across all splits/folds.

[6]:
```
queue_multiclass.metricsAggregate_df(
    selected_metrics = None
    , selected_stats = None
    , sort_by        = 'predictor_id'
    , ascending      = True
).head(12)
```

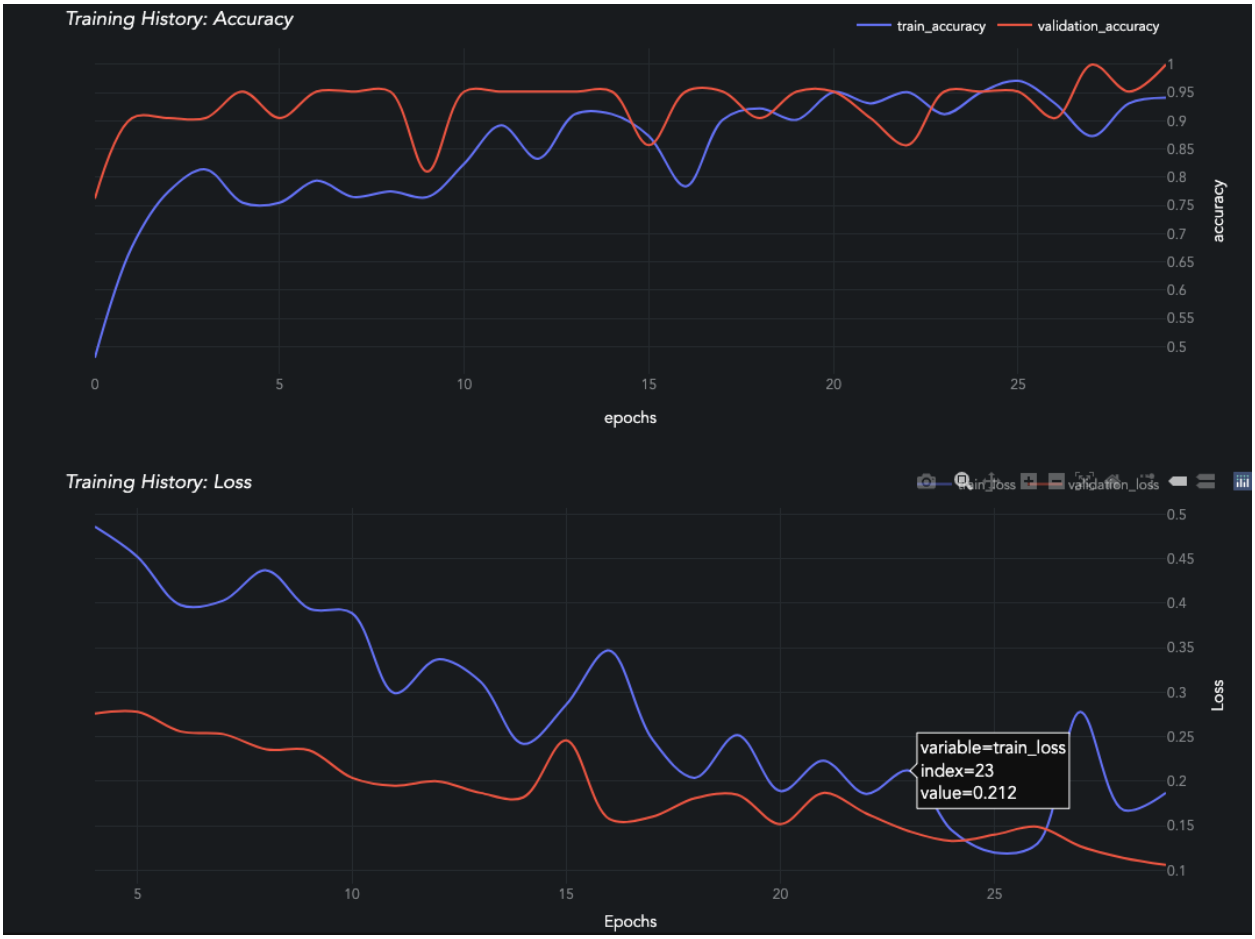
[6]:
```
   hyperparamcombo_id  job_id  predictor_id     metric  maximum  minimum  \
0                  17      23            25   accuracy    0.963    0.810
1                  17      23            25         f1    0.963    0.806
2                  17      23            25       loss    0.317    0.240
3                  17      23            25  precision    0.967    0.822
4                  17      23            25     recall    0.963    0.810
5                  17      23            25    roc_auc    1.000    0.966

     pstdev  median      mean
0  0.063608   0.912  0.895000
1  0.065301   0.911  0.893333
2  0.031633   0.271  0.276000
3  0.060139   0.917  0.902000
4  0.063608   0.912  0.895000
5  0.013880   0.983  0.983000
```

6.3.3 Job Visualization

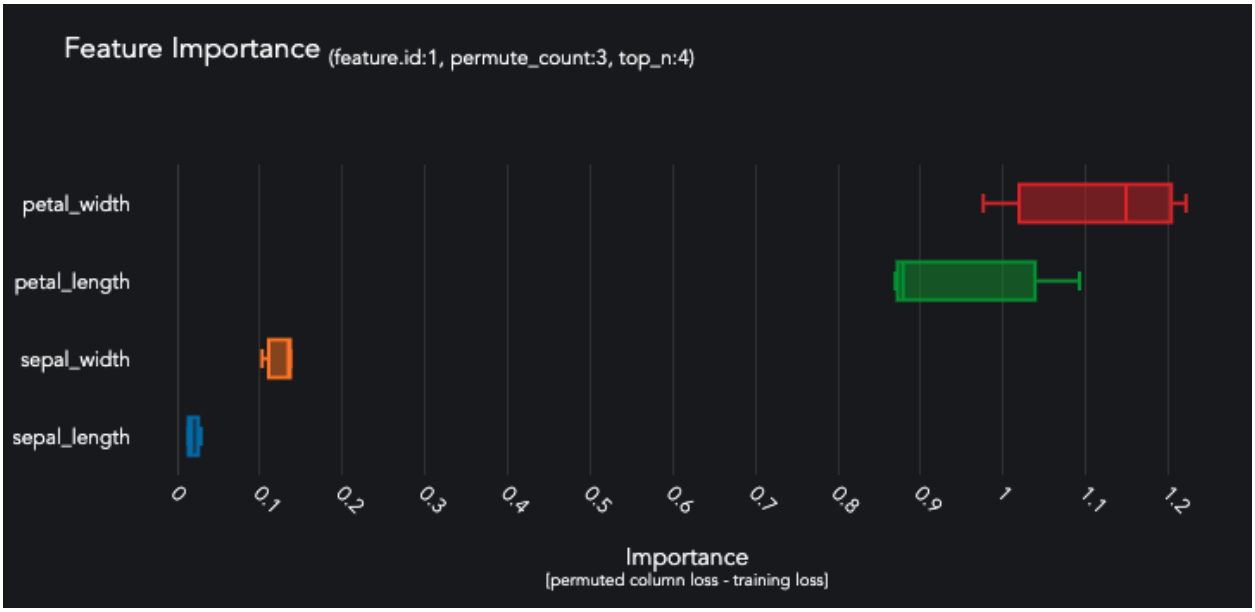
A learning curve will be generated for each train-evaluation pair of metrics in the `Predictor.history` dictionary. Reference the low-level API for more details.

Loss values in the first few epochs can often be extremely high before they plummet and become more gradual. This really stretches out the graph and makes it hard to see if the evaluation set is diverging or not. The `skip_head:bool` parameter skips displaying the first 15% of epochs so that figure is easier to interpret.

[ ]:
```
queue_multiclass.jobs[0].predictors[0].plot_learning_curve(skip_head=True)
```




```
[ ]: queue_multiclass.jobs[0].predictors[0].predictions[0].plot_feature_importance(top_n=4)
```

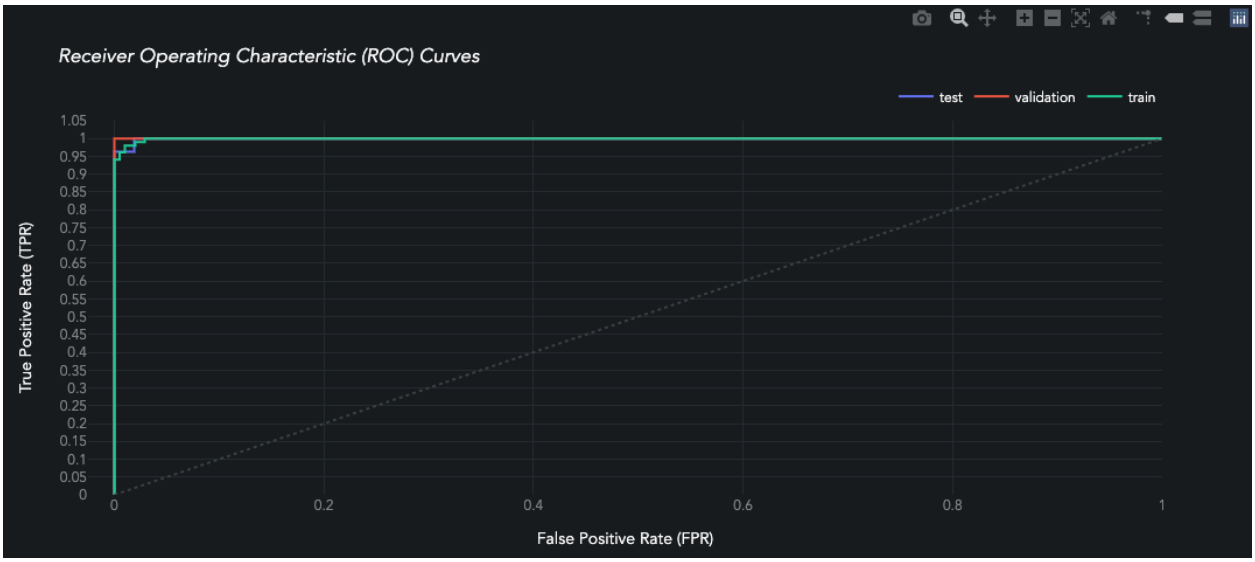


These classification metrics are preformatted for plotting.

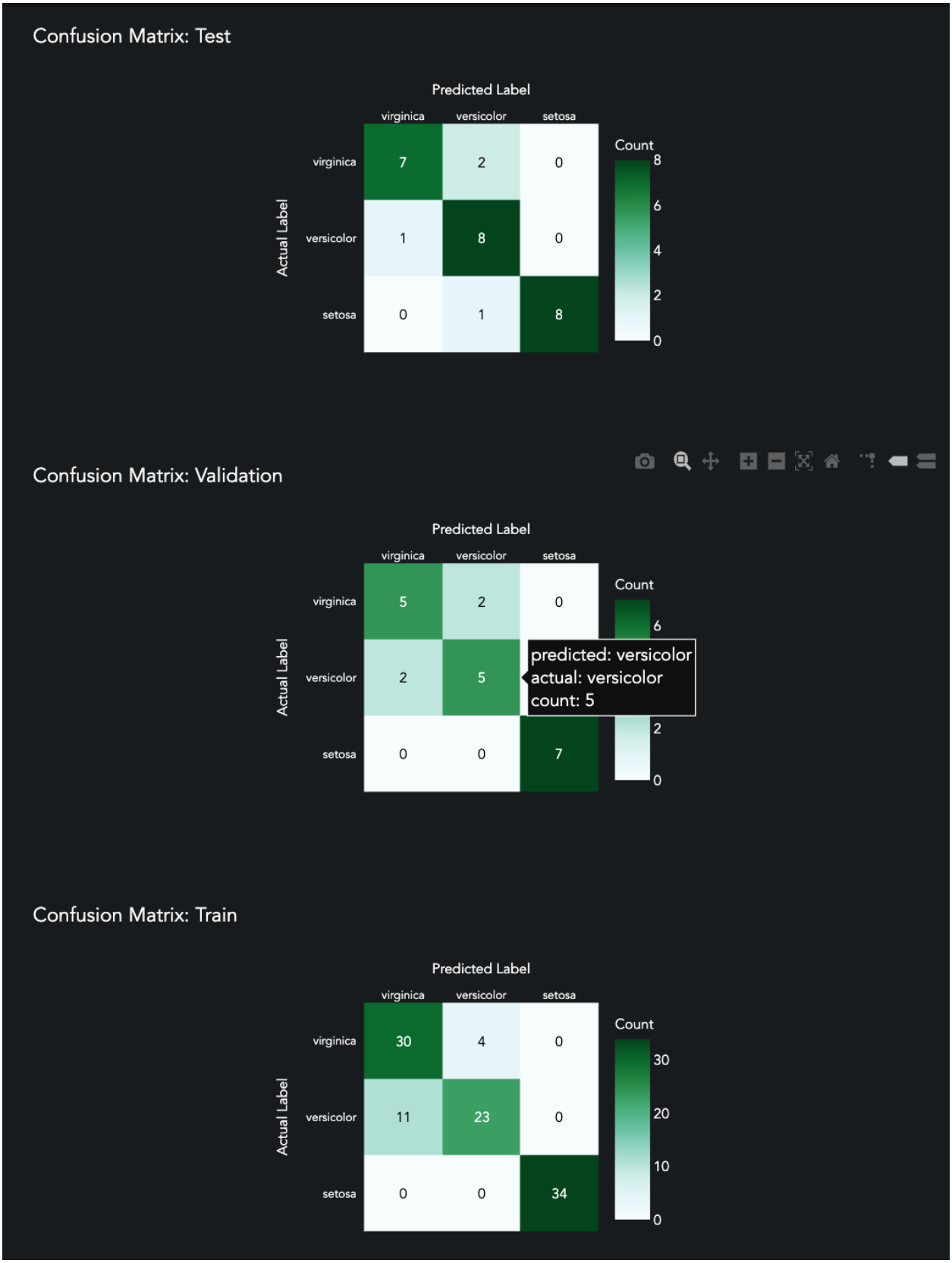
[9]:
```
queue_multiclass.jobs[0].predictors[0].predictions[0].plot_data['test'].keys()
```

[9]:
```
dict_keys(['confusion_matrix', 'roc_curve', 'precision_recall_curve'])
```

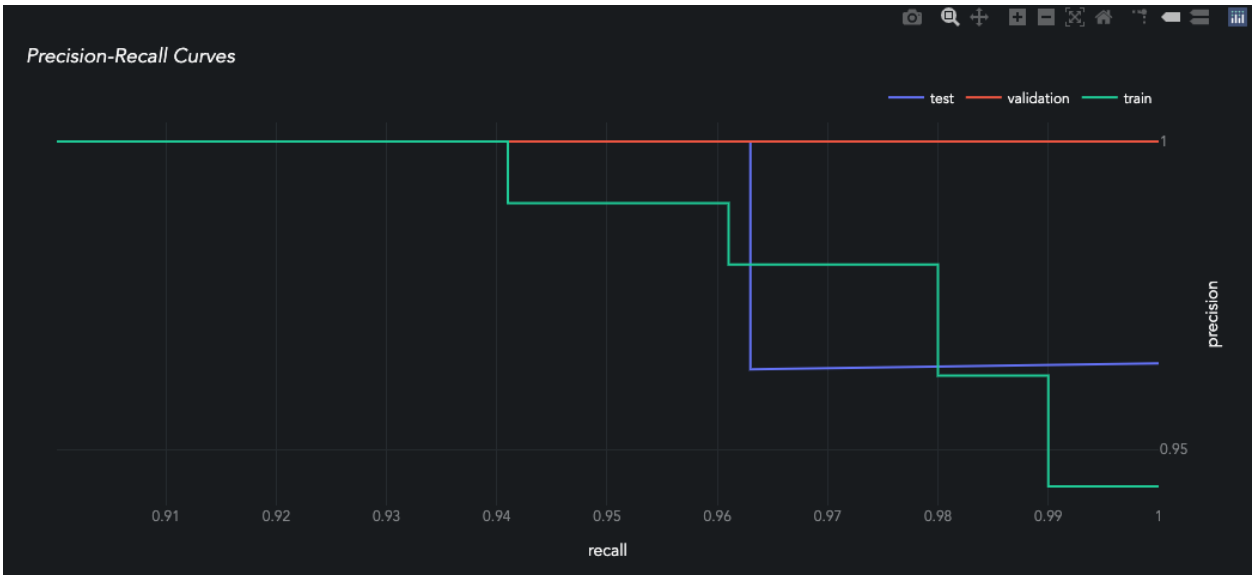
[ ]:
```
queue_multiclass.jobs[0].predictors[0].predictions[0].plot_roc_curve()
```



[ ]:
```
queue_multiclass.jobs[0].predictors[0].predictions[0].plot_confusion_matrix()
```


Confusion Matrix: Test
Predicted Label
virginica
versicolor
setosa
Actual Label
virginica
versicolor
setosa
7
2
0
1
8
0
0
1
8
Count
8
6
4
2
0
Confusion Matrix: Validation
Predicted Label
virginica
versicolor
setosa
Actual Label
virginica
versicolor
setosa
5
2
0
2
5
0
0
7
predicted: versicolor
actual: versicolor
count: 5
Count
6
2
0
Confusion Matrix: Train
Predicted Label
virginica
versicolor
setosa
Actual Label
virginica
versicolor
setosa
30
4
0
11
23
0
0
0
34
Count
30
20
10
0

```
[ ]: queue_multiclass.jobs[0].predictors[0].predictions[0].plot_precision_recall()
```



6.3.4 Job Metrics

Each training `Prediction` contains the following metrics by split/fold:

```
[13]: from pprint import pprint as p
```

```
[14]: p(queue_multiclass.jobs[0].predictors[0].predictions[0].metrics)
```

```
{'test': {'accuracy': 0.963,
          'f1': 0.963,
          'loss': 0.24,
          'precision': 0.967,
          'recall': 0.963,
          'roc_auc': 1.0},
 'train': {'accuracy': 0.912,
           'f1': 0.911,
           'loss': 0.271,
           'precision': 0.917,
           'recall': 0.912,
           'roc_auc': 0.983},
 'validation': {'accuracy': 0.81,
                'f1': 0.806,
                'loss': 0.317,
                'precision': 0.822,
                'recall': 0.81,
                'roc_auc': 0.966}}
```

It also contains per-epoch `History` metrics calculated during model training.

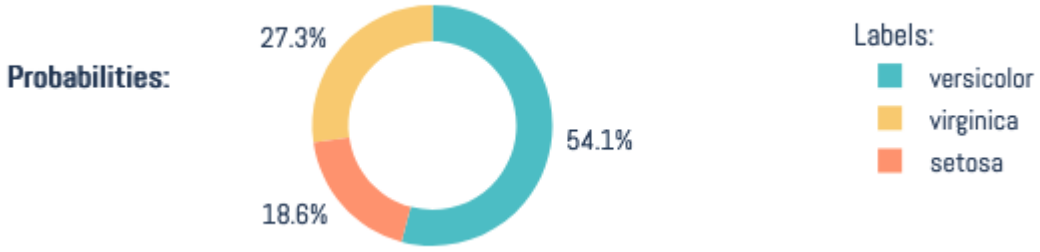
```
[15]: queue_multiclass.jobs[0].predictors[0].history.keys()
```

```
[15]: dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

6.3.5 Prediction Visualization

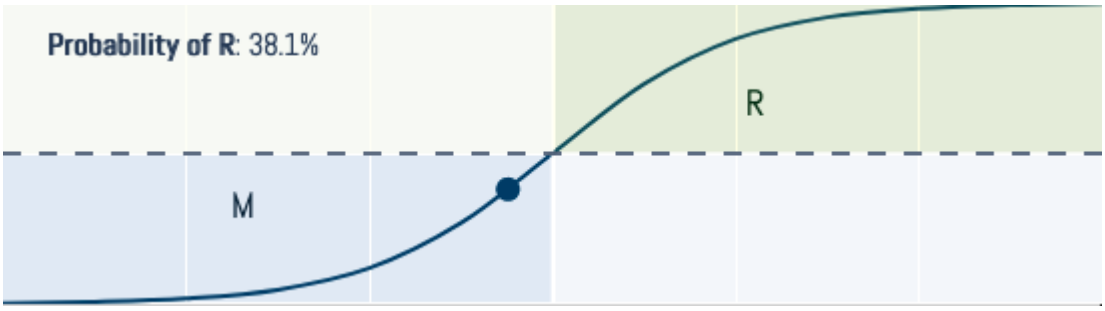
Multi-Label Classification Probabilities

```
[ ]: queue_multiclass.jobs[0].predictors[0].predictions[0].plot_confidence(prediction_index=0)
```



Binary Classification Probabilities

Also served by `plot_confidence()` for binary models.



6.3.6 Prediction Metrics

```
[5]: queue_multiclass.jobs[0].predictors[0].predictions[0].probabilities['train'][0]
[5]: array([0.98889786, 0.0101095 , 0.00099256], dtype=float32)
```

6.4 Regression

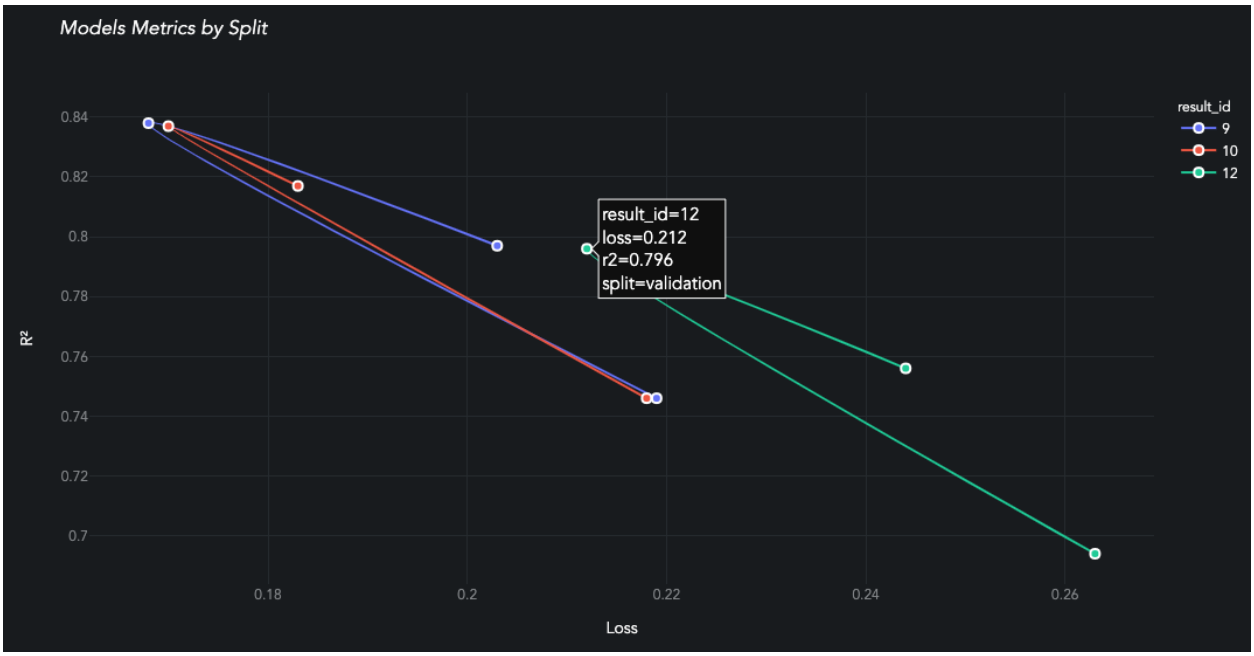
Let's quickly generate a trained quantification model to inspect.

```
[18]: %%capture
queue_regression = tests.tf_reg_tab.make_queue()
queue_regression.run_jobs()
```

6.4.1 Queue Visualization

When evaluating a regression-based `Queue.analysis_type`, the following `score_type:str` are available: r2, mse, and explained_variance.

```
[ ]: queue_regression.plot_performance(
        max_loss=1.5, score_type='r2', min_score=0.65
    )
```



6.4.2 Queue Metrics

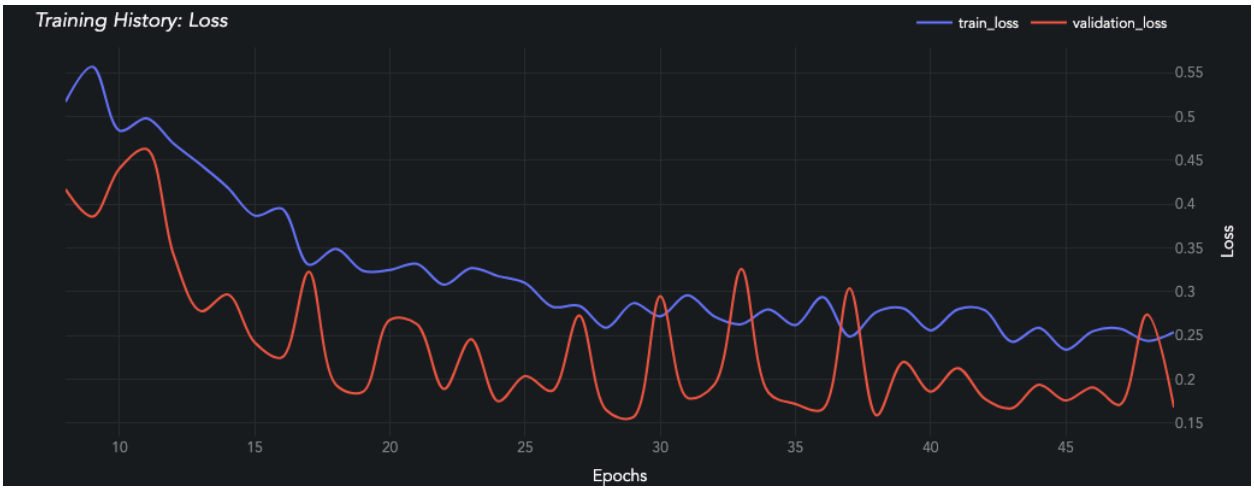
```
[ ]: queue_regression.metrics_df().head(9)
```

These are also aggregated by metric across all splits/folds.

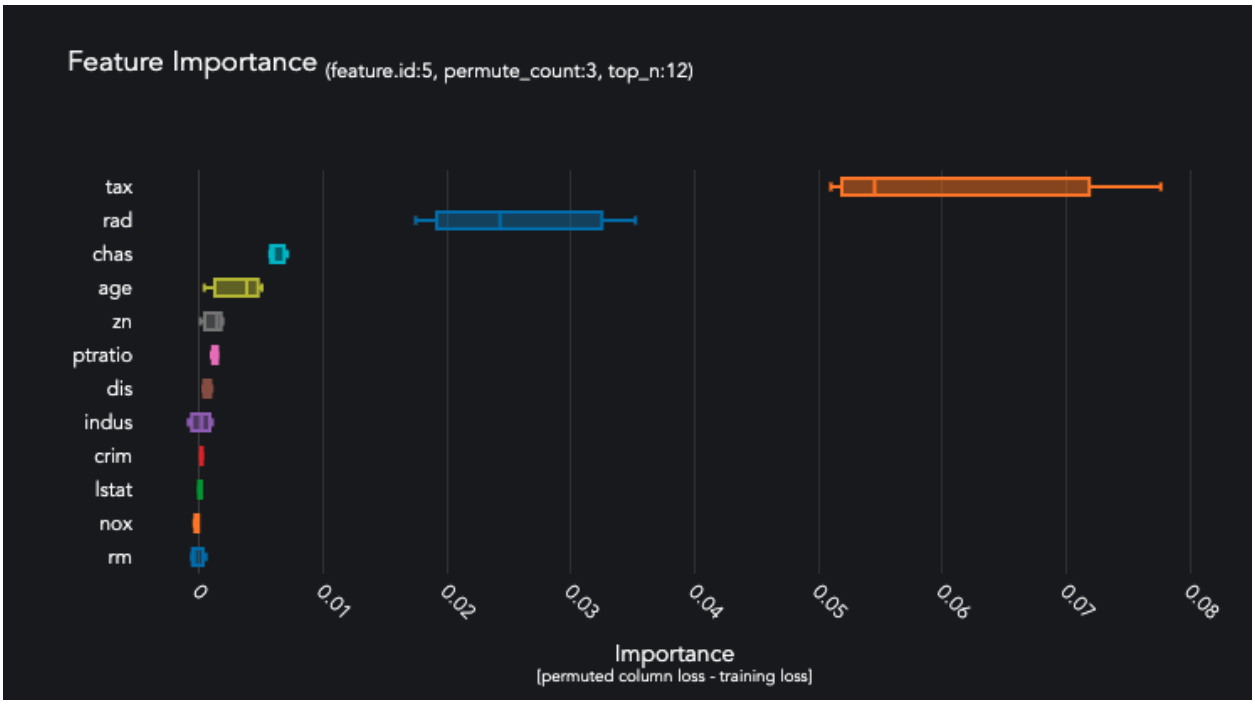
```
[ ]: queue_regression.metricsAggregate_df().tail(12)
```

6.4.3 Job Visualization

```
[ ]: queue_regression.jobs[0].predictors[0].plot_learning_curve(skip_head=True)
```




```
[ ]: queue_regression.jobs[0].predictors[0].predictions[0].plot_feature_importance(top_n=12)
```



6.4.4 Job Metrics

Each training `Prediction` contains the following metrics.

[19]:
```
p(queue_regression.jobs[0].predictors[0].predictions[0].metrics)
```

```
{'test': {'explained_variance': 0.048,
          'loss': 0.754,
          'mse': 1.045,
          'r2': -0.045},
 'train': {'explained_variance': 0.036,
           'loss': 0.733,
           'mse': 0.971,
           'r2': 0.029},
 'validation': {'explained_variance': 0.048,
                'loss': 0.678,
                'mse': 0.822,
                'r2': 0.043}}
```

It also contains per-epoch metrics calculated during model training.

[20]:
```
queue_regression.jobs[0].predictors[0].history.keys()
```

[20]:
```
dict_keys(['loss', 'mean_squared_error', 'val_loss', 'val_mean_squared_error'])
```

CHAPTER

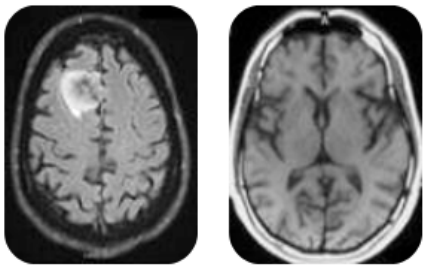
SEVEN

DEEP LEARNING 101

Boiling down a neural network to its fundamental concepts.

Generative	Given what we know about rows 1:1000 → generate row 1001.
Discriminative	Given what we know about columns A:F → determine column G's values.

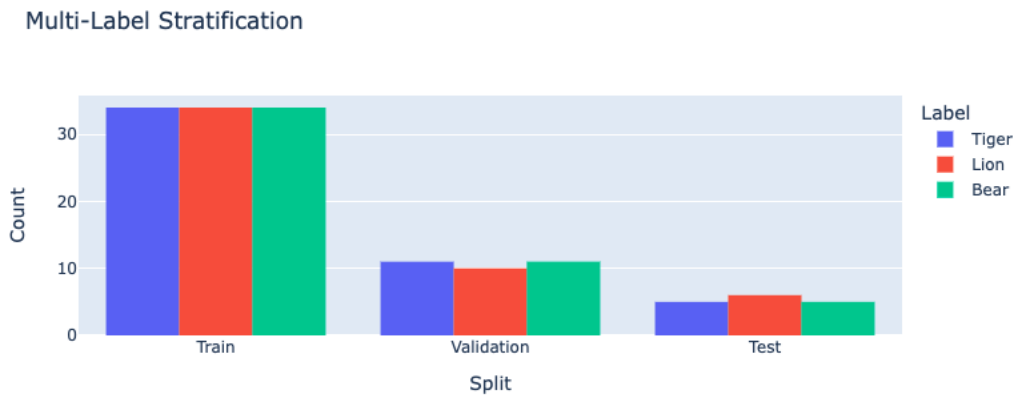
Categorize	What is it?	aka classify	e.g. benign vs malignant? which species?
Quantify	How much?	aka regress	e.g. price? distance? age? radioactivity?



Binary	Checking for the presence of a single condition	e.g. tumor
Multi-Label	When there are many possible outcomes	e.g. species

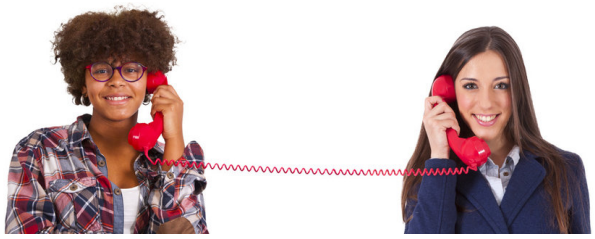
---

Features	Indepent Variable *(X)*	Informative columns like *num_legs*, *has_wings*, *has_shell*.
Label	Dependent Variable *(y)*	The *species* column that we want to predict.

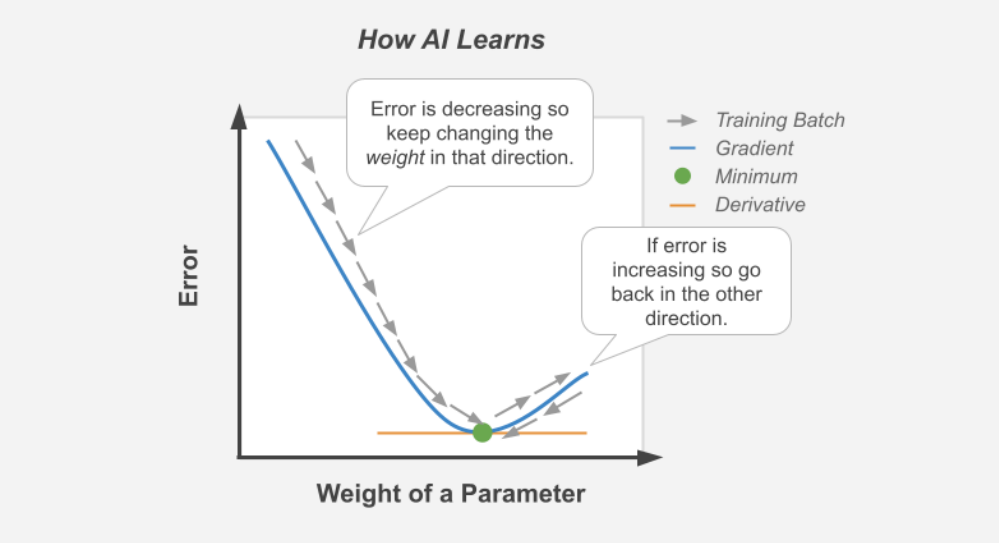


Train	67%	What the algorithm is trained on/ learns from.
Validation	21%	What the model is evaluated against during training.
Test	12%	Blind *holdout* for evaluating the model at the end of training.

Binarize	Categorical	1 means presence, 0 means absence.
OneHotEncode(OHE)	Categorical	Expand 1 multi-category col into many binary cols.
Ordinal	Categorical	[Bad form] Each category assigned an integer [0,1,2].
Scale	Continuous	Shrink the range of values between -1:1 or 0:1.

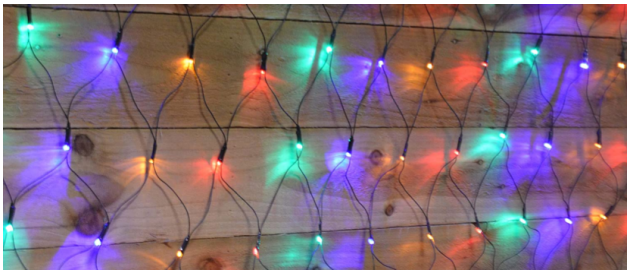


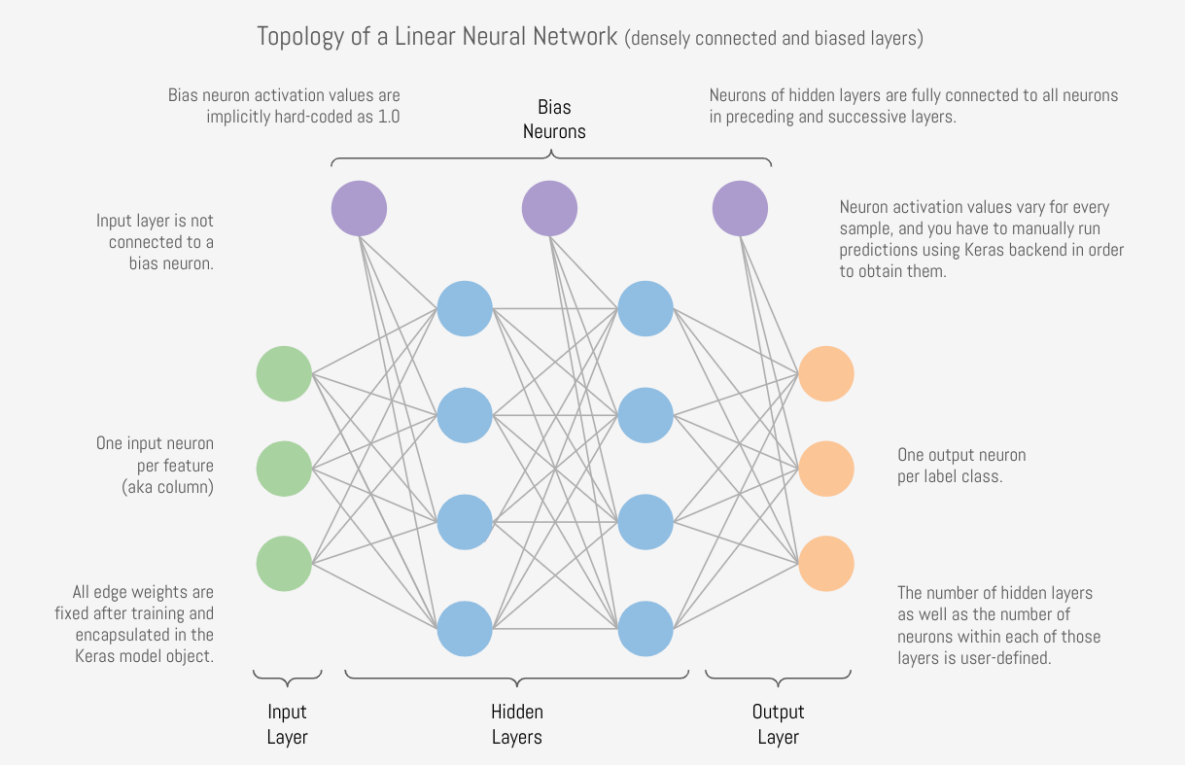
```
species = (num_legs * x) + (has_wings * y) + (has_shell * z)
```



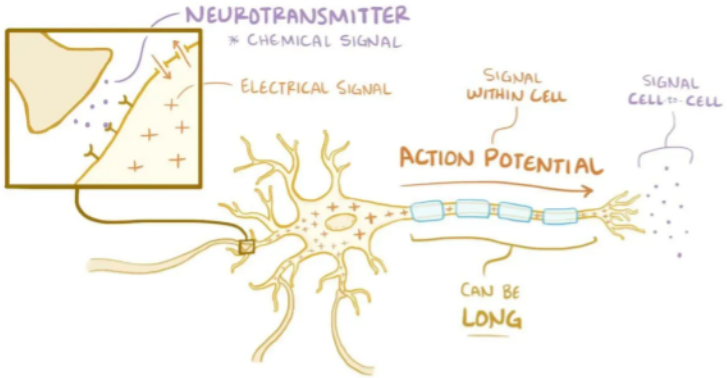
Linear	Tabular	e.g. spreadsheets & tables.
Convolutional	Positional	e.g. images, videos, & networks.
Recurrent	Ordered	e.g. time, text, & DNA.

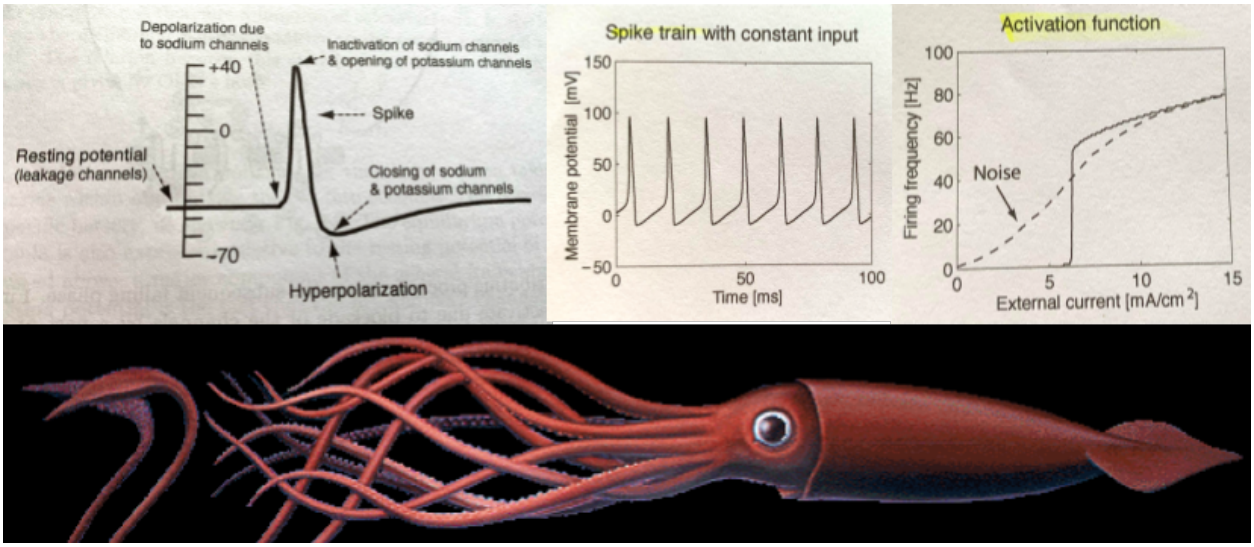
Nodes	neurons	participants in the network	e.g. lightbulbs
Edges	weights	connect (aka link) the nodes together	e.g. wires

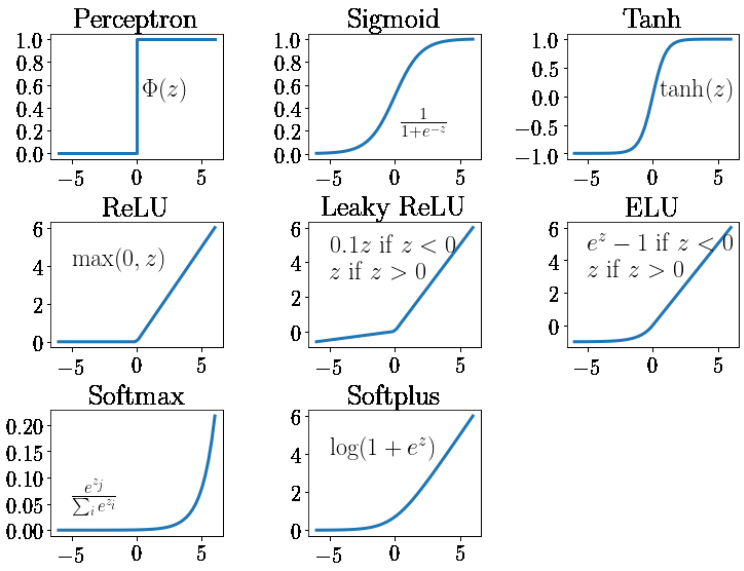




Input	Receives the data. Mirrors the shape of incoming features.
Hidden	Learns from patterns in the features. The number of layers & neurons based on feature complexity.
Output	Compares predictions to the real label. Mirrors shape of labels (# of categories).
Regulatory	[Not pictured here] *Dropout, BatchNorm, MaxPool* keep the network balanced and help prevent overfitting.


NEUROTRANSMITTER
* CHEMICAL SIGNAL
ELECTRICAL SIGNAL
SIGNAL WITHIN CELL
SIGNAL CELL-TO-CELL
ACTION POTENTIAL
CAN BE LONG

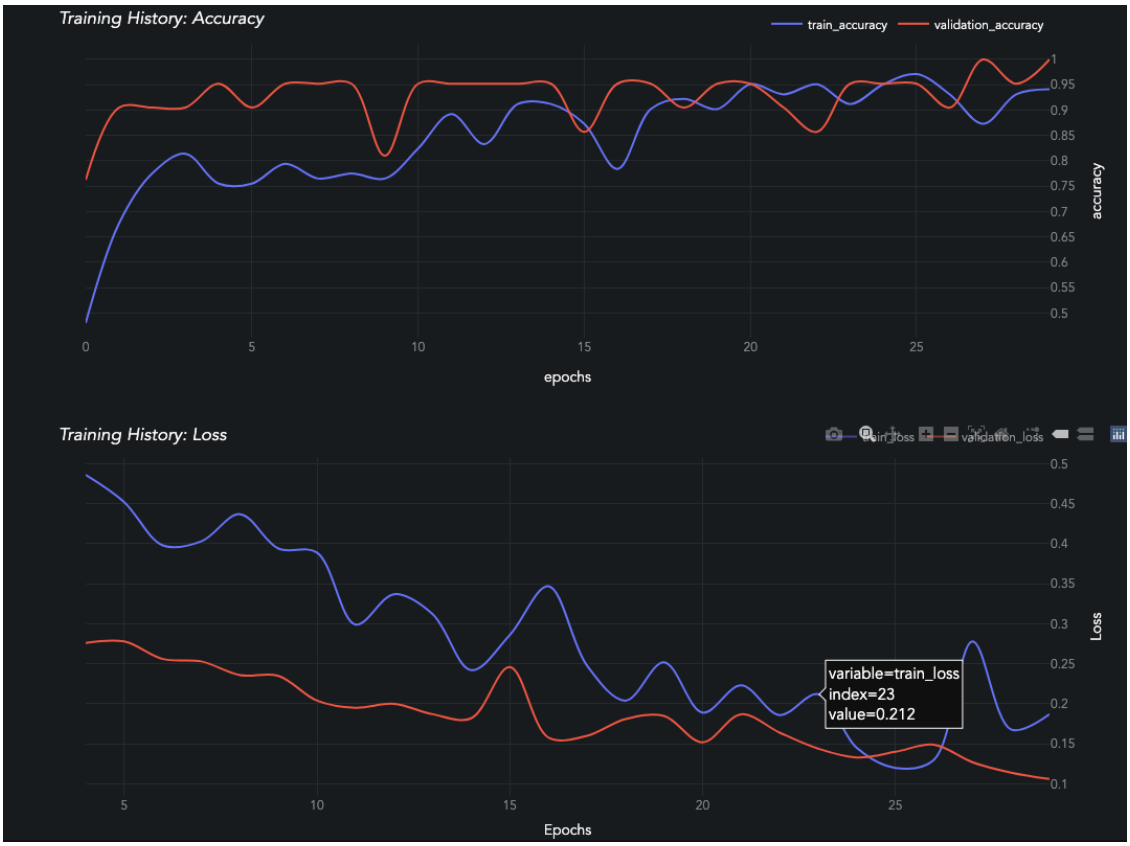

Depolarization due to sodium channels
Inactivation of sodium channels & opening of potassium channels
+40
0
-70
Spike
Resting potential (leakage channels)
Closing of sodium & potassium channels
Hyperpolarization
Spike train with constant input
Membrane potential [mV]
Time [ms]
Activation function
Firing frequency [Hz]
Noise
External current [mA/cm²]


Perceptron
$\Phi(z)$
Sigmoid
$\frac{1}{1+e^{-z}}$
Tanh
$\tanh(z)$
ReLU
$\max(0, z)$
Leaky ReLU
$0.1z$ if $z < 0$
z if $z > 0$
ELU
$e^z - 1$ if $z < 0$
z if $z > 0$
Softmax
$\frac{e^{z_j}}{\sum_i e^{z_i}}$
Softplus
$\log(1 + e^z)$

Input	In a linear network, the receiving layer does not have an activation function.
Hidden	The de facto activation function is *ReLU*. Rarely, *Tanh*.
Output	*Sigmoid* for binary classify. *Softmax* for multi-label classify. None for regression.

BinaryCrossentropy	Binary classification.
CategoricalCrossentropy	Multi-label classification.
MeanSquaredError or MeanAbsoluteError.	Used for regression.

Accuracy	Classification.
R^2	Regression.



Duration	Food isn't fully cooked? Train for more *epochs* or decrease the size of each *batch*.
Parameters	Burning? Turn down *learning rate*. Tastes bad? Try *initialization*/ *activation* spices.
Topology	If the food doesn't fit in the pan, switch to a larger pan with deeper/ taller *layers*.
Regulation	Overfitting on the same old recipes? Add more *Dropout* to mix things up.


THIS IS YOUR MACHINE LEARNING SYSTEM?
YUP! YOU POUR THE DATA INTO THIS BIG PILE OF LINEAR ALGEBRA, THEN COLLECT THE ANSWERS ON THE OTHER SIDE.
WHAT IF THE ANSWERS ARE WRONG?
JUST STIR THE PILE UNTIL THEY START LOOKING RIGHT.
DATA
ANSWERS

CHAPTER

EIGHT

OPEN SOURCE

8.1 Purpose

The AIQC framework brings rapid & reproducible deep learning to open science. We strive to empower researchers with a free tool that is easy to integrate into their experiments. You can learn more about our mission here.

Our initial goal is to build a guided framework for each major type of data (tabular, image, sequence, text, graph) and analysis (classify, quantify, generate).

Ultimately, we'd like to create/ incorporate domain-specific preprocessing pipelines, pre-trained models, and visualizations for each major scientific domain in order to accelerate discovery in each field of science.

8.2 How can I get involved?

- Create a post on the discussion board and introduce yourself so that we can help get you up to speed!
  - If you tell us what topics you are interested in, then we can help you get in sync with the project in a way that is enjoyable for everyone.
  - If you want to join the community calls, then be sure to include your timezone and email in your introduction.
- Jump into the conversation in the Slack group.

8.3 How can I contribute?

- Have a look at the GitHub Issues for something that interests you.
  - Keep an eye out for issues are tagged with *good first issue*.
  - We can design a *sprint* for you that represents a meaningful contribution to the project. This is not limited to software engineering. For example, it could be something like graphic design, blog-writing, or grant-writing. As described in the *Governance* section, completing a *sprint* is how you join the Core Team.
- Take a look at the Pull Request Template.
  - This document provides a PR checklist and shows how to run the tests.
  - We'll review your PR and provide comments on how to get it ready for production. You can also converse with us in the Slack channel mentioned below.

8.4 Setting up dev environment

Have a read through the Installation section of the documentation for information about OS, Python versions, and optional Jupyter extensions.

Here is how you can clone the source code, install dependencies, and environment:

```
git clone git@github.com:aiqc/AIQC.git

cd AIQC

pip install --upgrade -r requirements_dev.txt
pip install --upgrade -r requirements.txt
pip uninstall aiqc -y

git checkout -b my_feature

python
>>> import aiqc
```

Before you begin developing, make sure that you do NOT have the *aiqc* package installed. This may be counterintuitive at first, but remember, you are building the package yourself. So if you imported the pip package, then you are running scripts against the pip package, not your source code.

Also, have a look at the Documentation's README for documentation building dependencies as well as some do's and don'ts.

8.5 Programming style

- Prioritize human readability, maintainability, and simplicity over conciseness, efficiency, and performance.
  - Do not over-optimize. Schemas change. Over-optimization can make it hard for others to understand, integrate, and adapt your code. It's better to move on to the next problem than making the current functionality *x%* faster.
  - Can you do it without lambda, function composition, or some complex 1-liner that takes someone else an hour to reverse engineer? Remember, most data scientists inherently aren't world class software engineers, and vice versa!
  - If the code is not used in multiple places, then do not make it a function just for the sake of it. It's better to read code top-to-bottom rather than reverse engineering a complex web of someone else's functions.
  - When in doubt, use many lines to express yourself, lots of whitespace, and shallow depth of modularity.
- When handling edge cases, apply the Pareto principle (80-20); try to handle obvious pitfalls, but don't make the program more complex than it has to be.
  - *Do* - verify that the file/directory exists when users provide a path argument, provide helpful error messages, and validate dtypes & shapes of input, but;
  - *Don't* - spend a month writing your own custom checksum handler or solution for Python multiprocessing on Windows. Again, move on to something else rather than chasing an asymptote. The edge case code you wrote may be so specific that it is hard to maintain.
- If in doubt, ask what other people think in a Discussion.

8.6 Code of conduct

Inspired by NumFOCUS leaders and 'Google I/O 2008 - Open Source Projects and Poisonous People'

- *Be cordial and welcoming*; Communities are living, breathing social organisms where we can all learn to better ourselves while coming together to enact meaningful change.
- *Agree to disagree*; on one hand, acknowledge the merits of the ideas of others and be willing to adapt your opinion based on new information, but, on the other hand, **do not** sacrifice what you truly believe in for the sake of consensus.
- *Help educate & mentor*; point people in the right direction to get started, but don't continue to help those who won't help themselves. Open source projects are a way for people to break out of their 9-5, so a lot of people are learning new things. In generally, be significantly less rigid than the StackOverflow community, but do ask people to state what they have tried, share their code, share their env, etc. Remember, AIQC is at the confluence of multiple disciplines, so err on the side of educating. English is also a 2nd language for many, so be patient.
- *Speaking about other technologies*; When you mention other tools, give them as much praise as you can for what they have done well. Don't shy away from our benefits, but do take care to phrase your comparison politely. You never know who you will get connected with. For example, "We wanted our tool to be persistent and easy-to-use because that's what it was going to take to get it into the hands of researchers. When we tried out other tools for ourselves as practicioners, we didn't feel like they fully satisfied our criteria."
- *Violations*; If you feel that certain behavior does not jive with the code of conduct, please report the instance to the community manager, Layne Sadler. In particular, any instance of either hate, harassment, or heinous prejudice will result in an immediate and permanent ban without the explicit need for a vote.

8.7 Guild bylaws (aka governance)

Based on advice from our friends at Django and Jupyter:

- "Governance in the early days was largely about reviewing PRs and asking ourselves, *'Should we do this?'*"
- "This is an unfortunate need, but you should have as part of it how someone can be removed from their role, voluntarily or otherwise."
- "In smaller projects, the leadership handles the quality of what's brought into the project's technical assets and shepherds the people."

The vernacular is modeled after a D&D-like guild in order to make governance less dry.

Band of Squires [aka Public Participants]:

- Anyone that participates in community chat/ discussion board or submits a PR, but has not yet completed a *sprint*.
- All are welcome. Get in touch and we will help design a *sprint* for you.
- PRs must be reviewed by a council member before a merger.
- All participants are subject to the *Code of Conduct*.

Fellowship of Archmages [aka Core Team]:

- Anyone who has completed 2 *sprints* (level I, II, III, IV).
- Participates in the biweekly team meetings.
- Helps administer the Slack community and discussion board.
- PRs must still be reviewed by a council member before a merger.
- If it becomes absolutely necessary, the team can submit a proposal to remove/demote a team member for either repeated breach of *Code of Conduct* (2 strike depending on severity) or technical malpractice (1 strike). The penalty may be either temporary or permanent depending on the severity.
- The team can force any proposal submitted to the discussion board up to the council with a 2/3 vote (assuming there are at least 3 people on the team). However, rational discourse is preferred to forced votes.

Council of Warlocks [aka Steering Committee]:

- Anyone who has completed 5+ *sprints* (level V+). With at least 2 sprints being related to core deep learning functionality.
- Ability to approve PRs.
- Ability to release software (e.g. PyPI).
- Design sprints for new members.
- Inclusion in the license copyright.
- The council can vote on proposals submitted to the discussion board regarding the technical direction/ architecture of the project. Decisions will be made by a 2/3 majority, using U.S. Senate as a precedent.
- The Grand Warlock [aka Project Creator] reserves the right to a tie-breaking vote. They can also veto a majority vote on a given proposal, and the proposal cannot be brought up again until 6 months have passed. After which, if the same proposal succeeds a vote a second time, then they cannot veto it.
- If it becomes absolutely necessary, the council can submit a proposal to remove/demote a team member for either repeated breach of *Code of Conduct* (2 strikes depending on severity), intentional malpractice (1 strike), technical incompetence (3 strikes). The penalty may be either temporary or permanent depending on the severity.
- Changes to either the *Governance*, *Code of Conduct*, or *License* require a proposal to the discussion board.

8.8 AIQC, Inc. is open core

All AIQC functionality developed to date is open source. However, for the following reasons, AIQC is incorporated and will adhere to an *open core* business model in the long run:

- Many successful open source projects have championed the open core model while managing to remain free:
  - Notable examples include: NumFOCUS JuliaLang - JuliaComputing, Apache Spark - Databricks, NumFOCUS Dask - Coiled & SaturnCloud, Apache Zeppelin - Zepl, Apache Kafka - Confluent, GridAI - PyTorch Lightning, Dash & Plotly - Plotly, MongoDB, RStudio.
  - It's analogous to the *freemium* days of web 2.0 and apps. 95% of people get access to the free service while 5% of users pay for the premium options that solve their specific problems.
- In order to apply for certain government grant programs like the National Science Foundation (NSF) and DARPA (creators of the internet), it is *required* to form a business entity. Both JuliaLang and Dask have seen great success with this path.
  - Unfortunately, many grant application processes are explicitly reserved for individuals that are affiliated with esteemed institutions, which makes them off limits for everyday citizens.
- In reality, the continued success of many open source projects, even those that are not directly associated with a company, depends upon both funding and salaried contributors coming from corporate sponsors with which they collaborate.
  - This assistance naturally comes with a degree influence, sometimes formally in the shape of governance roles. Forming your own company to help financially back the project helps the project creators have an equal seat at the table of sponsors.
- In practice, when collaborating with large research institutes or R&D teams, they typically need: technical support to get up and running, consulting to help it fit their use cases, or they want to evaluate the technology on their data through a trial consulting engagement.
- The most prominent AI labs, like OpenAI and DeepMind, have been able to champion open research in a corporate setting. That's also where the best deep learning talent is going.
- The Global Alliance for Genomics & Health (GA4GH)] eventually had to organize for legal protection.
- Many biotech businesses offer either free or reduced pricing for students and academics as a healthy compromise.
- To paraphrase Isaacson's, The Innovators,: *"The first computer that was invented is sitting in a university basement in Iowa gathering dust. However, the 2nd computer was manufactured by IBM. You could find it on every professional desktop and point-of-sale counter in the world. It led the digital revolution."*

8.9 Open source

8.9.1 Choosing a license

AIQC is made open source under the Berkeley Software Distribution (BSD) 3-Clause license. This license is approved by the Open Source Initiative (OSI), which is preferred by NumFOCUS. 3-Clause BSD is used by notable projects including: NumPy, Scikit-learn, Dask, Matplotlib, IPython, and Jupyter.

BSD is seen as a *permissive* license, as opposed to *restrictive*. The major implications are that people that incorporate AIQC into their work are *neither* obligated to release their source code as open source, nor restricted to publishing their work under the same license.

> The simplest argument for AIQC adopting the BSD license is that AIQC uses upstream BSD projects. Therefore, it should pay it forward by using the same license and allowing others the same freedom it enjoys.
>
> On one hand, the permissive nature of this license means that the cloud providers can fork this project and release it as their own closed source cloud service, which has been a recurring theme [a, b, etc.]. On the other hand, feedback from our friends in the Python community was that people would avoid using libraries with restrictive licenses, like AGPL, in their work. They explained that they aren't allowed to open source their work and they "don't want to get their legal team involved." This begs the question, what good is being open source under a restrictive license if no one can *actually* use your software? Hopefully the cloud providers will put programs in place to contribute either code or profit (similar to the classic App Store revenue-sharing model) back to the communities whose projects they fork.
>
> Consideration of 4-Clause BSD; The *original* BSD license included an additional *advertising clause* that states: "All advertising materials mentioning features or use of this software must display the following acknowledgement: This product includes software developed by [...]." Which helps, in part, to address the widespread complaint of, "If you are going to fork our project, at least give us a nod." We've actually seen this play out at Datto. The company used software written by StorageCraft and Oracle for years, and eventually they ended up adding a StorageCraft badge to their marketing collateral. It felt fair. However, the *advertising clause* of 4-Clause BSD made it officially incompatible with GPL-licensed projects and, in practice, 3-Clause BSD projects! The latter is the deciding factor. If we want to be a part of a BSD-based community, then we cannot hinder it.

The copyright section is modeled after the IPython project.

Disclaimer; We still need to investigate BSD 3-Clause Clear and Apache 2.0 regarding patent & trademark rights.

CHAPTER

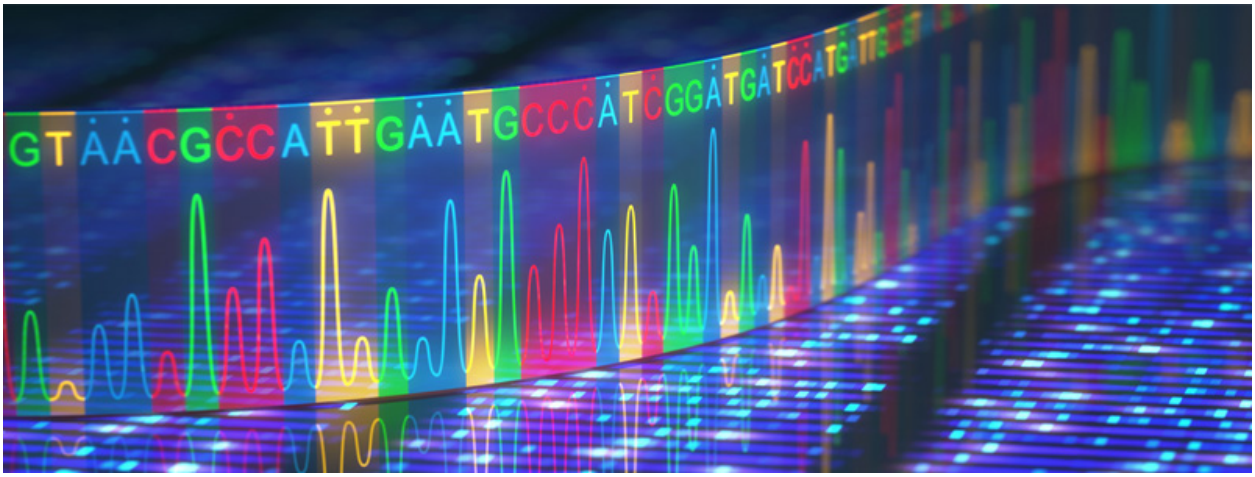
NINE

COMPETITION

CHAPTER

TEN

MISSION



10.1 Why Does AIQC Exist?

Over the past 4 years, I worked with the top 5 pharmaceutical companies to analyze national biobanks, such as the UK Biobank and Genomics Medicine Ireland, for the genomic-drivers of complex diseases.

In the face of such challenging & important problems, I was shocked that big pharma's primary form of analysis was the basic statistical test known as an association study, which dates back to the Victorian era. I kept expecting someone to say, *"Okay, now is the time for us to start using deep learning,"* but it never happened. If the researchers at the most well-financed companies in the world weren't equipped to take advantage of AI, then how would it ever be possible for non-profit scientists?

Deep learning has the power to accelerate the rate of scientific discovery by acting as a torch that reveals the laws of nature through data-driven pattern recognition. When it comes to global crises like combatting pandemics and reversing climate catastrophe, the human race is at a point where it needs to make major scientific advances over a short period of time in order to survive. So let's empower our smartest people with the best analytical tools we have.

10.2 1. Accelerate science by making deep learning accessible.

- Reduce the amount of programming and data science know-how required to perform deep learning. This unattainable skillset trifecta causes machine learning to be underutilized in science. What would Newton & Einstein have discovered with the power of deep learning?
- Provide field-specific deep learning solutions for research in the form of: pipelines for preprocessing scientific file formats, pre-trained models for transfer learning, and visualizations of predictions.

10.3 2. Bring the scientific method to data science.

- Make machine learning less of a black box by implementing "Quality control (QC)" protocols comprised of best practice validation rules.
- Reproducibly record not only the machine learning experiments, but also the lineage for preparing data. This is important for combatting bias during the data gathering and evaluation phases.

10.4 3. Break down walled gardens to keep science open.

- This toolset provides research teams a standardized method for ML-based evidence, as opposed to each research team cobbling together their own approach. An AIQC file should be submitted alongside publications and model zoo entries as a proof.
- The majority of research doesn't happen in the cloud, it's performed on the personal computers of individuals. We empower the non-cloud researchers: the academic/ institute HPCers, the remote server SSH'ers, and everyday laptop warriors.
- If the entire scientific community does not have access to the toolset used to conduct the experiment, then it is not reproducible.

- Kennedy - Peace; our survival demands unified, systematic action.
- Kennedy - Moon; lead the advancement of science for the good of mankind.


AIQC
Prepare Data
Register datasets
Stratify samples
Preprocess data
Track Experiments
Train models
Tune parameters
Evaluate models
Gain Insight
Rank features
Make predictions
Decode results
Python package
SQLite database
User interface

CHAPTER

ELEVEN

AIQC