RDataLoader

ROOT 6.40+ | Python 3.9+ | experimental

IMPORT AND CONSTRUCT

```
from ROOT.Experimental.ML import RDataLoader

dl = RDataLoader(rdf, batch_size=64, target="y")
```

CONSTRUCTOR ARGUENTS

rdataframes	RDF|list	one or more rdataframes
batch_size	int	rows per batch
batches_in_memory	int	shuffle buffer size
columns	list[str]	batches to read
max_vec_sizes	dict	max length per Rvec column
vec_padding	float	pad velue for short Rvecs
target	str|list	label column(s)
weights	str	event weight column
shuffle	bool	shuffle flag
drop_remainder	bool	drop last incomplete batch
set_seed	int	RNG seed
load_eager	bool	Load full dataset into RAM
sampling_type	str	"unders/oversampling"
sampling_ratio	float	minority/majority ratio
replacement	bool	undersampling with replacement

PROPERTIES

.columns	all loaded columnnames
.train_columns	feature columns (excl. target & weights)
.target_columns	target column name(s)
.num_batches	batches per epoch (after first iteration)

TWO LOADING MODES

Lazy loading Data is read chunk by chunk from disk. Low memory footprint, suitable for any dataset size.

```
dl = RDataLoader(rdf, batch_size=512)  # default
```

Eager loading Full dataset loaded in memory upfront. Better for small dataset iterated many times.

```
dl = RDataLoader(rdf, batch_size=512, load_eager=True)
```

RESAMPLING

Correct class imbalance by oversampling the minority or undersampling the majority.

Requires two RDataFrames and load_eager=True.

```
dl = RDataLoader(
    [rdf_sig, rdf_bkg],
    load_eager=True,
    sampling_type="oversampling",
    sampling_ratio=1.0,
)
```

TRAIN / TEST SPLIT

Split your dataset into training and testing subsets:

```
train, test = dl.train_test_split(test_size=0.2)
```

Need train / val / test? Call it twice on two separate loaders built from the same RDataFrame:

```
# split off 20% for test
train_val, test = dl.train_test_split(test_size=0.2)

# split the remaining 80% into train / val
# 0.125 × 0.8 = 10% of total
train, val = train_val.train_test_split(test_size=0.125)
```

VECTOR COLUMNS

Variable-length branches are flattened into fixed-width columns: declare the max size per branch, shorter vectors are padded.

```
dl = RDataLoader(
    rdf,
    columns=["jets_pt", "label"],
    max_vec_sizes={"jets_pt": 10:},
    vec_padding=0.0,
    target="label",
)
```

ITERATING BATCHES

as_torch Yields torch.Tensor batches.

```
# no target (reconstruction)
for X in dl.as_torch():
    loss = loss_fn(model(X), X)

# with target (classification)
for X, y in dl.as_torch():
    loss = loss_fn(model(X), y)

# on GPU
for X, y in dl.as_torch(device="cuda"):
    loss = loss_fn(model(X), y)

# with weights
for X, y, w in dl.as_torch():
    loss = (loss_fn(model(X), y) * w).mean()
```

as_tensorflow Yields tf.data.Dataset batches.

```
# repeat dataset
ds = dl.as_tensorflow().repeat(n_epochs)

# with target → model.fit
model.fit(ds, epochs=10)
```

as_numpy Yields numpy array batches.

```
for X, y in dl.as_numpy():
    print(X.shape)
```

HINTS

- **batches_in_memory ↑** = better randomisation, higher memory use
- **batches_in_memory ↓** = smaller memory use, limited shuffle
- **load_eager=True** eliminates per-epoch I/O overhead; best when the dataset fits in RAM and you train for many epochs
- **set_seed** sets the RNG to a fixed integer for reproducible train/val splits and epoch shuffling
- **shuffle=False** gives a deterministic order, useful for debugging or when order carries meaning