**Dynamic Feature/ Channel Selection**

import os

import torch

import numpy as np

import pandas as pd

from torch.utils.data import Subset, TensorDataset

GITHUB\_URL = 'https://raw.githubusercontent.com/iancovert/dynamic-selection/main/datasets/'

def data\_split(dataset, val\_portion=0.2, test\_portion=0.2, random\_state=0):

'''

Split dataset into train, val, test.

Args:

dataset: PyTorch dataset object.

val\_portion: percentage of samples for validation.

test\_portion: percentage of samples for testing.

random\_state: random seed.

'''

# Shuffle sample indices.

rng = np.random.default\_rng(random\_state)

inds = np.arange(len(dataset))

rng.shuffle(inds)

# Assign indices to splits.

n\_val = int(val\_portion \* len(dataset))

n\_test = int(test\_portion \* len(dataset))

test\_inds = inds[:n\_test]

val\_inds = inds[n\_test:(n\_test + n\_val)]

train\_inds = inds[(n\_test + n\_val):]

# Create split datasets.

test\_dataset = Subset(dataset, test\_inds)

val\_dataset = Subset(dataset, val\_inds)

train\_dataset = Subset(dataset, train\_inds)

return train\_dataset, val\_dataset, test\_dataset

def get\_xy(dataset):

'''

Extract inputs (x) and outputs (y) from dataset object.

Args:

dataset: PyTorch dataset object.

'''

x, y = zip(\*list(dataset))

if isinstance(x[0], np.ndarray):

return torch.from\_numpy(np.array(x)), torch.from\_numpy(np.array(y))

elif isinstance(x[0], torch.Tensor):

if isinstance(y[0], (int, float)):

return torch.stack(x), torch.tensor(y)

else:

return torch.stack(x), torch.stack(y)

else:

raise ValueError(f'not sure how to concatenate data type: {type(x[0])}')

def load\_spam(features=None):

# Load data.

data\_dir = os.path.join(GITHUB\_URL, 'spam.csv')

data = pd.read\_csv(data\_dir)

# Set features.

if features is None:

features = np.array([f for f in data.columns if f not in ['Outcome']])

else:

assert 'Outcome' not in features

if isinstance(features, list):

features = np.array(features)

# Extract x, y.

x = np.array(data.drop(['Outcome'], axis=1)[features]).astype('float32')

y = np.array(data['Outcome']).astype('int64')

# Create dataset object.

dataset = TensorDataset(torch.from\_numpy(x), torch.from\_numpy(y))

dataset.features = features

dataset.input\_size = x.shape[1]

dataset.output\_size = len(np.unique(y))

return dataset

def load\_diabetes(features=None):

# Load data.

data\_dir = os.path.join(GITHUB\_URL, 'diabetes.csv')

data = pd.read\_csv(data\_dir)

# Set features.

if features is None:

features = np.array([f for f in data.columns if f not in ['Outcome']])

else:

assert 'Outcome' not in features

if isinstance(features, list):

features = np.array(features)

# Extract x, y.

x = np.array(data.drop(['Outcome'], axis=1)[features]).astype('float32')

y = np.array(data['Outcome']).astype('int64')

# Create dataset object.

dataset = TensorDataset(torch.from\_numpy(x), torch.from\_numpy(y))

dataset.features = features

dataset.input\_size = x.shape[1]

dataset.output\_size = len(np.unique(y))

return dataset

def load\_miniboone(features=None):

# Load data.

data\_dir = os.path.join(GITHUB\_URL, 'miniboone.csv')

data = pd.read\_csv(data\_dir)

# Set features.

if features is None:

features = np.array([f for f in data.columns if f not in ['Outcome']])

else:

assert 'Outcome' not in features

if isinstance(features, list):

features = np.array(features)

# Extract x, y.

x = np.array(data.drop(['Outcome'], axis=1)[features]).astype('float32')

y = np.array(data['Outcome']).astype('int64')

# Create dataset object.

dataset = TensorDataset(torch.from\_numpy(x), torch.from\_numpy(y))

dataset.features = features

dataset.input\_size = x.shape[1]

dataset.output\_size = len(np.unique(y))

return dataset