# pylint: disable=arguments-differ,no-member,too-many-statements

''' Contains HashedPercentileDiscretizerCalibrator used for calibration '''

from .percentile\_discretizer import PercentileDiscretizerCalibrator

import numpy as np

import twml

class HashingDiscretizerCalibrator(PercentileDiscretizerCalibrator):

''' Accumulates features and their respective values for HashingDiscretizer calibration.

This calibrator perfoms the same actions as PercentileDiscretizerCalibrator but it's

`to\_layer` method returns a HashingDiscretizer instead.

'''

def \_create\_discretizer\_layer(self, n\_feature, hash\_map\_keys, hash\_map\_values,

feature\_offsets, name):

# Need to sort hash\_map\_keys according to hash\_map\_values

# just in case they're not in order of being put in the dict

# hash\_map\_values is already 0 through len(hash\_map\_values)-1

hash\_map\_keys = hash\_map\_keys.flatten()

# why is this float32 in PercentileDiscretizerCalibrator.to\_layer ????

# need int for indexing

hash\_map\_values = hash\_map\_values.flatten().astype(np.int32)

feature\_ids = np.zeros((len(hash\_map\_keys),), dtype=np.int64)

for idx in range(len(hash\_map\_keys)):

feature\_ids[hash\_map\_values[idx]] = hash\_map\_keys[idx]

return twml.contrib.layers.HashingDiscretizer(

feature\_ids=feature\_ids,

bin\_vals=self.\_bin\_vals.flatten(),

n\_bin=self.\_n\_bin + 1, # (self.\_n\_bin + 1) bin\_vals for each feature\_id

out\_bits=self.\_out\_bits,

cost\_per\_unit=500,

name=name

)