from .hashing\_utils import make\_feature\_id

from twml.contrib.layers.hashing\_discretizer import HashingDiscretizer

import numpy as np

class TFModelDiscretizerBuilder(object):

def \_\_init\_\_(self, num\_bits):

self.num\_bits = num\_bits

def build(self, tf\_model\_initializer):

'''

:param tf\_model\_initializer: dictionary of the following format:

{

"features": {

"bias": 0.0,

"binary": {

# (feature name : feature weight) pairs

"feature\_name\_1": 0.0,

...

"feature\_nameN": 0.0

},

"discretized": {

# (feature name : index aligned lists of bin\_boundaries and weights

"feature\_name\_1": {

"bin\_boundaries": [1, ..., inf],

"weights": [0.0, ..., 0.0]

}

...

"feature\_name\_K": {

"bin\_boundaries": [1, ..., inf],

"weights": [0.0, ..., 0.0]

}

}

}

}

:return: a HashingDiscretizer instance.

'''

discretized\_features = tf\_model\_initializer["features"]["discretized"]

max\_bins = 0

feature\_ids = []

bin\_vals = []

for feature\_name in discretized\_features:

bin\_boundaries = discretized\_features[feature\_name]["bin\_boundaries"]

feature\_id = make\_feature\_id(feature\_name, self.num\_bits)

feature\_ids.append(feature\_id)

np\_bin\_boundaries = [np.float(bin\_boundary) for bin\_boundary in bin\_boundaries]

bin\_vals.append(np\_bin\_boundaries)

max\_bins = max(max\_bins, len(np\_bin\_boundaries))

feature\_ids\_np = np.array(feature\_ids)

bin\_vals\_np = np.array(bin\_vals).flatten()

return HashingDiscretizer(

feature\_ids=feature\_ids\_np,

bin\_vals=bin\_vals\_np,

n\_bin=max\_bins,

out\_bits=self.num\_bits

)