import re

from twitter.deepbird.io.util import \_get\_feature\_id

class Parser(object):

def parse(self, line):

match = re.search(self.pattern(), line)

if match:

return self.\_parse\_match(match)

return None

def pattern(self):

raise NotImplementedError

def \_parse\_match(self, match):

raise NotImplementedError

class BiasParser(Parser):

'''

Parses the bias feature available in lolly model tsv files.

'''

def pattern(self):

'''

Matches lines like:

unified\_engagement bias -0.935945

:return: a RegEx that extracts feature weight.

'''

return r"\t(bias)\t([^\s]+)"

def \_parse\_match(self, match):

return float(match.group(2))

class BinaryFeatureParser(Parser):

'''

Parses binary features available in lolly model tsv files.

'''

def pattern(self):

'''

Matches lines like:

unified\_engagement encoded\_tweet\_features.is\_user\_spam\_flag -0.181130

:return: a RegEx that extracts feature name and weight.

'''

return r"\t([\w\.]+)\t([^\s]+)"

def \_parse\_match(self, match):

return (match.group(1), float(match.group(2)))

class DiscretizedFeatureParser(Parser):

'''

Parses discretized features available in lolly model tsv files.

'''

def pattern(self):

'''

Matches lines like:

unified\_engagement encoded\_tweet\_features.user\_reputation.dz/dz\_model=mdl/dz\_range=1.000000e+00\_2.000000e+00 0.031004

:return: a RegEx that extracts feature name, bin boundaries and weight.

'''

return r"([\w\.]+)\.dz\/dz\_model=mdl\/dz\_range=([^\s]+)\t([^\s]+)"

def \_parse\_match(self, match):

left\_bin\_side, right\_bin\_side = [float(number) for number in match.group(2).split("\_")]

return (

match.group(1),

left\_bin\_side,

right\_bin\_side,

float(match.group(3))

)

class LollyModelFeaturesParser(Parser):

def \_\_init\_\_(self, bias\_parser=BiasParser(), binary\_feature\_parser=BinaryFeatureParser(), discretized\_feature\_parser=DiscretizedFeatureParser()):

self.\_bias\_parser = bias\_parser

self.\_binary\_feature\_parser = binary\_feature\_parser

self.\_discretized\_feature\_parser = discretized\_feature\_parser

def parse(self, lolly\_model\_reader):

parsed\_features = {

"bias": None,

"binary": {},

"discretized": {}

}

def process\_line\_fn(line):

bias\_parser\_result = self.\_bias\_parser.parse(line)

if bias\_parser\_result:

parsed\_features["bias"] = bias\_parser\_result

return

binary\_feature\_parser\_result = self.\_binary\_feature\_parser.parse(line)

if binary\_feature\_parser\_result:

name, value = binary\_feature\_parser\_result

parsed\_features["binary"][name] = value

return

discretized\_feature\_parser\_result = self.\_discretized\_feature\_parser.parse(line)

if discretized\_feature\_parser\_result:

name, left\_bin, right\_bin, weight = discretized\_feature\_parser\_result

discretized\_features = parsed\_features["discretized"]

if name not in discretized\_features:

discretized\_features[name] = []

discretized\_features[name].append((left\_bin, right\_bin, weight))

lolly\_model\_reader.read(process\_line\_fn)

return parsed\_features

class DBv2DataExampleParser(Parser):

'''

Parses data records printed by the DBv2 train.py build\_graph function.

Format: [[dbv2 logit]][[logged lolly logit]][[space separated feature ids]][[space separated feature values]]

'''

def \_\_init\_\_(self, lolly\_model\_reader, lolly\_model\_features\_parser=LollyModelFeaturesParser()):

self.features = lolly\_model\_features\_parser.parse(lolly\_model\_reader)

self.feature\_name\_by\_dbv2\_id = {}

for feature\_name in list(self.features["binary"].keys()) + list(self.features["discretized"].keys()):

self.feature\_name\_by\_dbv2\_id[str(\_get\_feature\_id(feature\_name))] = feature\_name

def pattern(self):

'''

:return: a RegEx that extracts dbv2 logit, logged lolly logit, feature ids and feature values.

'''

return r"\[\[([\w\.\-]+)\]\]\[\[([\w\.\-]+)\]\]\[\[([\w\.\- ]+)\]\]\[\[([\w\. ]+)\]\]"

def \_parse\_match(self, match):

feature\_ids = match.group(3).split(" ")

feature\_values = match.group(4).split(" ")

value\_by\_feature\_name = {}

for index in range(len(feature\_ids)):

feature\_id = feature\_ids[index]

if feature\_id not in self.feature\_name\_by\_dbv2\_id:

print("Missing feature with id: " + str(feature\_id))

continue

value\_by\_feature\_name[self.feature\_name\_by\_dbv2\_id[feature\_id]] = float(feature\_values[index])

return value\_by\_feature\_name