## 任务二解决算法

from numpy import loadtxt

from xgboost import XGBClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score

import pandas as pd

import numpy as np

import itertools

import networkx as nx

# load data

terror=pd.read\_csv('/Users/alibaba/gitlab/2018-mathmodels/q2/Q2\_1.csv',encoding='ISO-8859-1')

print(terror[:4])

# 提取第二问的所有特征列，并转换成 list

gname = terror['gname\_10encode'].as\_matrix()

print(gname[:10])

# seperate train and test

# test: 2015-2016, gname = 'unknown'

# train: 除了test 集的都是。考虑是否拿一点出来做validation。

# cleaned\_dataset 第一列是event\_id, 倒数第一列是goup\_id。

cleaned\_dataset = terror.as\_matrix()

cleaned\_dataset = cleaned\_dataset[cleaned\_dataset[:,0].argsort()]

print(type(cleaned\_dataset))

row = len(cleaned\_dataset)

cl = len(cleaned\_dataset[0])

for idx, i in enumerate(cleaned\_dataset):

cleaned\_dataset[idx][0] = idx

print(cleaned\_dataset[-1])

print(len(cleaned\_dataset))

# 新建一列 test ： 1表示属于test集， 0 表示属于train

append\_ar = np.zeros((row,1), dtype='int64')

for idx, r in enumerate(cleaned\_dataset):

# 先拿出2015-2016

# 再拿出unknown

if (r[2] == 2015) or (r[2] == 2016):

#gname = unkown =>>'0000000001'

if r[-1] == 1.0:

append\_ar[idx] = 1 # 是测试样本

else:

append\_ar[idx] = 0 # 2015-2016 间的已知gname，是训练样本

elif r[-1] == 1.0:

append\_ar[idx] = 11 # 其他年份的unknown类型， 可能是训练样本，也可舍弃。

else:

append\_ar[idx] = 12 # 其他年份的已知gname ,是训练样本。

# 在最后一列添加test标签

append\_dataset = np.append(cleaned\_dataset, append\_ar,axis =1)

# 提取测试样本

test = np.array( [r[:-1] for r in append\_dataset if r[-1] == 1 ])

num\_test = len(test)

print(test[-10])

# 提取训练样本

train = np.array( [r[:-1] for r in append\_dataset if r[-1] != 1])

num\_train = len(train)

print('feature dim %s' % len(train[0]))

print('train num is %s'% num\_train)

print('test num is %s' % num\_test)

print('total sum is %s '% row)

print(num\_test + num\_train)

# 提取不包含 ‘unknow’ 类型的训练样本

train\_no\_unknown = np.array( [r[:-1] for r in append\_dataset if (r[-1] == 0 or r[-1] == 12)])

num\_train\_no\_unknown = len(train\_no\_unknown)

# to generate groups

# row: group\_id, line: event\_id

def gen\_groups(dataset):

# dataset is a feature matrix , for each row, row[1] is eventid, row[0] is gname

# return sorted gname, row[0] is gname, row[1] is eventid

# return group\_ids

print(dataset[:10])

groups = dataset[dataset[:,0].argsort()]

ngroup = []

group\_1 = [row[1] for row in groups if row[0] == 1]

print(group\_1)

if int(len(group\_1)) != 0:

ngroup.append(group\_1)

#print(group\_1)

group\_id = 2

temp = []

tag = int(len(group\_1))

print(tag)

for row in groups[tag:]:

#print(row[0])

if row[0] == group\_id:

temp.append(row[1])

#print(temp)

elif len(temp) != 1:

#print(temp)

ngroup.append(temp)

group\_id = group\_id + 1

temp = [row[1]]

else:

#print('temp empty when group\_id = %d' % group\_id)

group\_id = group\_id + 1

temp = []

print('final goup\_id is %d' % group\_id)

return ngroup

train\_groups = gen\_groups(train\_no\_unknown[:,[-1,0]])

print(train\_groups[0])

print(len(train\_groups))

# generate train dataset with pos\_pairs, neg\_pairs and labels

# asume we have got group\_id and member list, we have a 2d list Groups, index is the groupid, row is all the eventid.

# for each group, generate negtive samples number

neg\_count = 5

def gen\_pairs(Groups, idx, neg\_count):

# to generate positive and negtive samples for the idx group

# neg\_percent: the number of negtive samples to generate = total negetive samples \* neg\_percent

pos\_pairs = [[x, y ] for idx\_x, x in enumerate(Groups[idx]) for idx\_y, y in enumerate(Groups[idx]) if idx\_x != idx\_y]

num\_pos = len(pos\_pairs)

pos\_labels = np.ones(num\_pos)

# randomly generate negtive samples

G = Groups

np.delete(G, (idx), axis=0)

neg\_pairs = []

flat\_G = [event for g in G for event in g]

for event in Groups[idx]:

np.random.shuffle(flat\_G)

neg\_data = flat\_G[:neg\_count]

neg\_pairs = neg\_pairs + [[event, neg\_event] for neg\_event in neg\_data]

neg\_labels = np.ones(len(neg\_pairs))

pairs = pos\_pairs + neg\_pairs

labels = [p for p in pos\_labels] + [n for n in neg\_labels]

return pairs, labels

def gen\_all\_pairs(Groups):

data = [gen\_pairs(Groups, idx, neg\_count) for idx,g in enumerate(Groups)]

Data\_pairs, Labels = zip(\*data)

return Data\_pairs, Labels

def gen\_train(Groups):

data = [gen\_pairs(Groups, idx, neg\_count) for idx,g in enumerate(Groups) if idx < 10]

print(data[:2])

train\_pairs, y\_train0 = zip(\*data)

print(len(train\_pairs))

print(len(train\_pairs[0]))

print(len(y\_train0))

print(train\_pairs[:2])

print(y\_train0[:2])

# concate the feature vector of two samples as one convated feature vector

X\_train1 = [cleaned\_dataset[int(pair[0])] + cleaned\_dataset[int(pair[1])] for pairs in train\_pairs for pair in pairs]

print('number of x\_train1 is %s' % len(X\_train1))

# 正反调换来一遍

X\_train2 = [np.append(pair[1], pair[0],axis =1) for pair in train\_pairs]

# 两个特征差分试一下

X\_train = X\_train1 #+ X\_train2

y\_train = [y for y in y\_train0 for yy in y] #+ y\_train0

return X\_train, y\_train

X\_train, y\_train = gen\_train(train\_groups)

# generate test dataset

def gen\_test\_pairs(fdata):

# fdata is the feature matrix of test data

# return concated feature, event\_id pairs

X\_test = []

test\_ids = []

for idx, event in enumerate(fdata):

test\_id = [[event[0],e[0]] for e in fdata[idx+1:]]

test\_ids = test\_ids + test\_id

test\_feature = [np.append(event, e, axis = 1) for e in fdata[idx+1:]]

X\_test = X\_test + test\_feature

return X\_test, test\_ids

X\_test, y\_test = gen\_train(test)

print('nnum of test is %d' % len(y\_test))

# xgboost//classifier

print(X\_train[:2])

print(y\_train[:2])

print(type(X\_train))

# fit model no training data

model = XGBClassifier()

model.fit(X\_train, y\_train)

print(model)

# make predictions for test data

y\_pred = model.predict(X\_test)

'''

predictions = [round(value) for value in y\_pred]

print(y\_pred[:100])

# evaluate predictions

accuracy = accuracy\_score(y\_test, predictions)

print("Accuracy: %.2f%%" % (accuracy \* 100.0))

'''

# grouping the y\_pred

import matplotlib.pyplot as plt

%matplotlib inline

G = nx.read\_edgelist('test\_g.edgelist', nodetype=int, create\_using=nx.DiGraph())

G = G.to\_undirected()

nx.draw(G)

g = nx.karate\_club\_graph()

fig, ax = plt.subplots(1, 1, figsize=(8, 6));

nx.draw\_networkx(g, ax=ax)