第一题

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from gensim import corpora, mods

import jieba.posseg as jp

from wordcloud import WordCloud as wc

import matplotlib as mpl

from sklearn import svm

from matplotlib import colors

# 环境准备

plt.style.use('seaborn')

sns.set(font\_scale=2)

plt.rcParams['font.sans-serif'] = ['SimHei']

plt.rcParams['axes.unicode\_minus'] = False

stopwords = ('根据', '图片', 'Moment', '是', '不', '也', '到', '将', '有', '都', 'G', '更', '就', '还', '会', '人', '要', '没有',

'可能', '需要', '就是', '包括', '已经', '还有', '甚至', '作为', '非常', '进行') # 停词

# 数据读取

def loadcsv():

f = open(r'附件1.csv', encoding='utf-8"')

df = pd.read\_csv(f, delimiter='\t', sep='\t', error\_bad\_lines=False)

df['char\_length'] = df['正文'].astype(str).apply(len)

return df

# jieba分词

def fenci(t):

print('开始分词')

flags = ('n', 'nr', 'ns', 'nt', 'eng', 'v', 'd') # 词性

# 分词

words\_ls = []

onesen = ""

for text in t:

words = [w.word for w in jp.cut(text) if w.flag in flags and w.word not in stopwords]

onesen += " ".join(words) + " "

words\_ls.append(words)

print('分词完毕')

return words\_ls, onesen

# lda 模型

def lda(words\_ls):

# 构造词典

dictionary = corpora.Dictionary(words\_ls)

print('词典构建完毕')

# 对每段文本依据词典生成向量

cor = [dictionary.doc2bow(words) for words in words\_ls]

print('lda模型开始训练')

lda = mods.ldamod.LdaMod(cor=cor, id2word=dictionary, num\_topics=3)

# 打印所有主题，每个主题显示 5 个词

for topic in lda.print\_topics(num\_words=10):

print(topic)

# 主题推断

print(lda.inference(cor))

print('lda模型已完成')

# 生成词云

def cloud(sen):

print('开始制作词云')

mywc = wc(font\_path="C:/Windows/Fonts/simkai.ttf", background\_color="white", max\_font\_s=130,

max\_words=200, stopwords=set(stopwords)).generate(sen)

image = mywc.to\_image()

mywc.to\_file('ciyun.png')

image.show()

print('词云制作结束')

# 支持向量机主题分类

def svm(x, y):

clf = svm.SVC(C=0.5, kernel='rbf', gamma=1, decision\_function\_shape='ovr')

clf.fit(x, y.ravel())

x1\_min, x1\_max = x[:, 0].min(), x[:, 0].max()

x2\_min, x2\_max = x[:, 1].min(), x[:, 1].max()

x1, x2 = np.mg[x1\_min:x1\_max:200j, x2\_min:x2\_max:200j]

g\_t = np.stack((x1.flat, x2.flat), axis=1)

g\_h = clf.predict(g\_t)

g\_h = g\_h.reshape(x1.shape)

cm\_l = mpl.colors.ListedColormap(['#A0FFA0', '#FFA0A0', '#A0A0FF'])

cm\_d = mpl.colors.ListedColormap(['g', 'b', 'r'])

plt.pcolormesh(x1, x2, g\_h, cmap=cm\_l)

plt.scatter(x[:, 0], x[:, 1], c=np.squeeze(y), edgecolor='k', s=50, cmap=cm\_d) # 样本点

plt.xlabel('关键词1的词频')

plt.ylabel('关键词2的词频')

plt.xlim(x1\_min, x1\_max)

plt.ylim(x2\_min, x2\_max)

plt.title('基于svm的主题分类示意图')

plt.grid()

plt.show()

df = loadcsv()

t = df['正文'].tolist()[0:5000]

wlist, sen = fenci(t)

lda(wlist)

cloud(sen)

svm(x,y)

第二题

import requests

import pandas as pd

import time

# 爬虫

def crawler():

re = []

header = {

'Content-Type': 'application/json; charset=utf-8',

'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/76.0.3809.100 Safari/537.36'

}

Cookie = {'Cookie': ''}

for ii in range(19):

# 抗疫日记

url\_base = 'https://m.weibo.cn/api/container/getIndex?containerid=100103type%3D60%26q%3' \

'D%23%E6%8A%97%E7%96%AB%E6%97%A5%E8%AE%B0%23%26t%3D0&page\_type=searchall'

url = url\_base + str(ii + 1)

html = requests.get(url, headers=header, cookies=Cookie)

try:

for jj in range(len(html.json()['data']['cards'])):

if html.json()['data']['cards'][jj]['mblog']['isLongText'] == False:

text=html.json()['data']['cards'][jj]['mblog']['text']

else:

text=html.json()['data']['cards'][jj]['mblog']['longText']['longTextContent']

fin=""

flag=False

for i in text:

if i=='<':

flag=True

if i=='>':

flag=False

if flag:

continue

fin+=i

data1 = [(html.json()['data']['cards'][jj]['mblog']['user']['id'], # 用户 ID

html.json()['data']['cards'][jj]['mblog']['user']['screen\_name'], # 用户名

html.json()['data']['cards'][jj]['mblog']['reposts\_count'], # 转发数

html.json()['data']['cards'][jj]['mblog']['comments\_count'], # 评论数

html.json()['data']['cards'][jj]['mblog']['attitudes\_count'], # 点赞数量

fin,

html.json()['data']['cards'][jj]['mblog']['cd\_at'], # 发表时间

html.json()['data']['cards'][jj]['mblog']['source'])] # 来源设备

data2 = pd.DataFrame(data1)

data2.to\_csv('weibo\_content-yq.csv', header=False, index=False, mode='a+',encoding='GBK')

re.extend(data1)

except:

print("抓取失败")

print('page ' + str(ii + 1) + ' has done')

time.sleep(3)

return re

comments = crawler()

第三题

import numpy as np,tensorflow as tf ,cPickle, random

from tensorflow.python.ops import tensor\_array\_ops, control\_flow\_ops

from dl import Gen\_D\_l, Dis\_dl

from ge import Ge

from dis import Dis

from r import R

from tar\_lstm import TAR\_LSTM

class Ge(object):

def \_\_init\_\_(self, n\_e, b\_s, e\_d, hid\_d,

seq\_len, s\_tok,

l\_rate=0.01, re\_g=0.95):

self.n\_e = n\_e

self.b\_s = b\_s

self.e\_d = e\_d

self.hid\_d = hid\_d

self.seq\_len = seq\_len

self.s\_tok = tf.constant([s\_tok] \* self.b\_s, dtype=tf.int32)

self.l\_rate = tf.Variable(float(l\_rate), tra=False)

self.re\_g = re\_g

self.g\_pa = []

self.d\_pa = []

self.t = 1.0

self.grad\_clip = 5.0

self.exp\_re = tf.Variable(tf.zeros([self.seq\_len]))

with tf.variable\_scope('ge'):

self.g\_eeddings = tf.Variable(self.init\_matrix([self.n\_e, self.e\_d]))

self.g\_pa.append(self.g\_eeddings)

self.g\_re = self.c\_re(self.g\_pa)

self.g\_o = self.c\_o(self.g\_pa)

self.x = tf.placeholder(tf.int32, shape=[self.b\_s,

self.seq\_len])

self.res = tf.placeholder(tf.float32, shape=[self.b\_s,

self.seq\_len])

with tf.device("/cpu:0"):

self.pr\_x = tf.transpose(tf.nn.eedding\_lookup(self.g\_eeddings, self.x),

perm=[1, 0, 2])

self.h0 = tf.zeros([self.b\_s, self.hid\_d])

self.h0 = tf.stack([self.h0, self.h0])

gen\_o = tensor\_array\_ops.TensorArray(dtype=tf.float32, s=self.seq\_len,

d\_s=False, infer\_shape=True)

gen\_x = tensor\_array\_ops.TensorArray(dtype=tf.int32, s=self.seq\_len,

d\_s=False, infer\_shape=True)

def \_g\_re(i, x\_t, h\_tm1, gen\_o, gen\_x):

h\_t = self.g\_re(x\_t, h\_tm1)

o\_t = self.g\_o(h\_t)

log\_prob = tf.log(tf.nn.softmax(o\_t))

n\_tok = tf.cast(tf.reshape(tf.multinomial(log\_prob, 1), [self.b\_s]), tf.int32)

x\_tp1 = tf.nn.eedding\_lookup(self.g\_eeddings, n\_tok) # b x e\_d

gen\_o = gen\_o.write(i, tf.reduce\_sum(tf.multiply(tf.one\_hot(n\_tok, self.n\_e, 1.0, 0.0),

tf.nn.softmax(o\_t)), 1))

gen\_x = gen\_x.write(i, n\_tok)

return i + 1, x\_tp1, h\_t, gen\_o, gen\_x

\_, \_, \_, self.gen\_o, self.gen\_x = control\_flow\_ops.while\_loop(

cond=lambda i, \_1, \_2, \_3, \_4: i < self.seq\_len,

body=\_g\_re,

loop\_vars=(tf.constant(0, dtype=tf.int32),

tf.nn.eedding\_lookup(self.g\_eeddings, self.s\_tok), self.h0, gen\_o, gen\_x))

self.gen\_x = self.gen\_x.stack()

self.gen\_x = tf.transpose(self.gen\_x, perm=[1, 0])

g\_pre = tensor\_array\_ops.TensorArray(

dtype=tf.float32, s=self.seq\_len,

d\_s=False, infer\_shape=True)

ta\_e\_x = tensor\_array\_ops.TensorArray(

dtype=tf.float32, s=self.seq\_len)

ta\_e\_x = ta\_e\_x.unstack(self.pr\_x)

def \_pre\_re(i, x\_t, h\_tm1, g\_pre):

h\_t = self.g\_re(x\_t, h\_tm1)

o\_t = self.g\_o(h\_t)

g\_pre = g\_pre.write(i, tf.nn.softmax(o\_t))

x\_tp1 = ta\_e\_x.read(i)

return i + 1, x\_tp1, h\_t, g\_pre

\_, \_, \_, self.g\_pre = control\_flow\_ops.while\_loop(

cond=lambda i, \_1, \_2, \_3: i < self.seq\_len,

body=\_pre\_re,

loop\_vars=(tf.constant(0, dtype=tf.int32),

tf.nn.eedding\_lookup(self.g\_eeddings, self.s\_tok),

self.h0, g\_pre))

self.g\_pre = tf.transpose(self.g\_pre.stack(),

perm=[1, 0, 2])

self.pre\_l = -tf.reduce\_sum(

tf.one\_hot(tf.to\_int32(tf.reshape(self.x, [-1])), self.n\_e, 1.0, 0.0) \* tf.log(

tf.clip\_by\_value(tf.reshape(self.g\_pre, [-1, self.n\_e]), 1e-20, 1.0)

)

) / (self.seq\_len \* self.b\_s)

pre\_opt = self.g\_optimizer(self.l\_rate)

self.pre\_grad, \_ = tf.clip\_by\_global\_norm(tf.gradients(self.pre\_l, self.g\_pa), self.grad\_clip)

self.pre\_updates = pre\_opt.apply\_gradients(zip(self.pre\_grad, self.g\_pa))

self.g\_l = -tf.reduce\_sum(

tf.reduce\_sum(

tf.one\_hot(tf.to\_int32(tf.reshape(self.x, [-1])), self.n\_e, 1.0, 0.0) \* tf.log(

tf.clip\_by\_value(tf.reshape(self.g\_pre, [-1, self.n\_e]), 1e-20, 1.0)

), 1) \* tf.reshape(self.res, [-1])

)

g\_opt = self.g\_optimizer(self.l\_rate)

self.g\_grad, \_ = tf.clip\_by\_global\_norm(tf.gradients(self.g\_l, self.g\_pa), self.grad\_clip)

self.g\_updates = g\_opt.apply\_gradients(zip(self.g\_grad, self.g\_pa))

def ge(self, sess):

os = sess.run(self.gen\_x)

return os

def pre\_step(self, sess, x):

os = sess.run([self.pre\_updates, self.pre\_l], f\_dict={self.x: x})

return os

def init\_matrix(self, shape):

return tf.random\_normal(shape, stddev=0.1)

def init\_vector(self, shape):

return tf.zeros(shape)

def c\_re(self, pa):

self.Wi = tf.Variable(self.init\_matrix([self.e\_d, self.hid\_d]))

self.Ui = tf.Variable(self.init\_matrix([self.hid\_d, self.hid\_d]))

self.bi = tf.Variable(self.init\_matrix([self.hid\_d]))

self.Wf = tf.Variable(self.init\_matrix([self.e\_d, self.hid\_d]))

self.Uf = tf.Variable(self.init\_matrix([self.hid\_d, self.hid\_d]))

self.bf = tf.Variable(self.init\_matrix([self.hid\_d]))

self.Wog = tf.Variable(self.init\_matrix([self.e\_d, self.hid\_d]))

self.Uog = tf.Variable(self.init\_matrix([self.hid\_d, self.hid\_d]))

self.bog = tf.Variable(self.init\_matrix([self.hid\_d]))

self.Wc = tf.Variable(self.init\_matrix([self.e\_d, self.hid\_d]))

self.Uc = tf.Variable(self.init\_matrix([self.hid\_d, self.hid\_d]))

self.bc = tf.Variable(self.init\_matrix([self.hid\_d]))

pa.extend([

self.Wi, self.Ui, self.bi,

self.Wf, self.Uf, self.bf,

self.Wog, self.Uog, self.bog,

self.Wc, self.Uc, self.bc])

def unit(x, hid\_memory\_tm1):

pre\_hid\_sta, c\_prev = tf.unstack(hid\_memory\_tm1)

i = tf.sigmoid(

tf.matmul(x, self.Wi) +

tf.matmul(pre\_hid\_sta, self.Ui) + self.bi

)

f = tf.sigmoid(

tf.matmul(x, self.Wf) +

tf.matmul(pre\_hid\_sta, self.Uf) + self.bf

)

o = tf.sigmoid(

tf.matmul(x, self.Wog) +

tf.matmul(pre\_hid\_sta, self.Uog) + self.bog

)

c\_ = tf.nn.tanh(

tf.matmul(x, self.Wc) +

tf.matmul(pre\_hid\_sta, self.Uc) + self.bc

)

c = f \* c\_prev + i \* c\_

current\_hid\_sta = o \* tf.nn.tanh(c)

return tf.stack([current\_hid\_sta, c])

return unit

def c\_o(self, pa):

self.Wo = tf.Variable(self.init\_matrix([self.hid\_d, self.n\_e]))

self.bo = tf.Variable(self.init\_matrix([self.n\_e]))

pa.extend([self.Wo, self.bo])

def unit(hid\_memory\_tuple):

hid\_sta, c\_prev = tf.unstack(hid\_memory\_tuple)

log = tf.matmul(hid\_sta, self.Wo) + self.bo

return log

return unit

def g\_optimizer(self, \*args, \*\*kwargs):

return tf.train.AdamOptimizer(\*args, \*\*kwargs)

E\_D = 32

HID\_D = 32

SEQ\_LEN = 20

S\_TOK = 0

PRE\_W\_N = 120

SEED = 88

B\_S = 64

dis\_eedding\_d = 64

dis\_f\_ss = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20]

dis\_n\_fs = [100, 200, 200, 200, 200, 100, 100, 100, 100, 100, 160, 160]

dis\_do\_keep\_prob = 0.75

dis\_l2\_reg\_lambda = 0.2

dis\_b\_s = 64

TOTAL\_B = 200

ged\_n = 10000

def ge\_s(sess, tra\_mod, b\_s, ged\_n, o\_file):

ged\_s = []

for \_ in range(int(ged\_n / b\_s)):

ged\_s.extend(tra\_mod.ge(sess))

def tar\_l(sess, tar\_lstm, d\_l):

nll = []

d\_l.re\_po()

for it in xrange(d\_l.n\_b):

b = d\_l.n\_b()

g\_l = sess.run(tar\_lstm.pre\_l, {tar\_lstm.x: b})

nll.append(g\_l)

return np.mean(nll)

def pre\_train\_w(sess, tra\_mod, d\_l):

sup\_g\_les = []

d\_l.re\_po()

for it in xrange(d\_l.n\_b):

b = d\_l.n\_b()

\_, g\_l = tra\_mod.pre\_step(sess, b)

sup\_g\_les.append(g\_l)

return np.mean(sup\_g\_les)

def main():

random.seed(SEED)

np.random.seed(SEED)

assert S\_TOK == 0

gen\_d\_l = Gen\_D\_l(B\_S)

like\_d\_l = Gen\_D\_l(B\_S)

vocab\_s = 5000

dis\_d\_l = Dis\_dl(B\_S)

ge = Ge(vocab\_s, B\_S, E\_D, HID\_D, SEQ\_LEN, S\_TOK)

tar\_pa = cPickle.load(open('s/tar\_pa.pkl'))

tar\_lstm = TAR\_LSTM(vocab\_s, B\_S, E\_D, HID\_D, SEQ\_LEN, S\_TOK,

tar\_pa)

dis = Dis(seq\_len=20, n\_classes=2, vocab\_s=vocab\_s,

eedding\_s=dis\_eedding\_d,

f\_ss=dis\_f\_ss, n\_fs=dis\_n\_fs,

l2\_reg\_lambda=dis\_l2\_reg\_lambda)

con = tf.ConProto()

con.gpu\_o.allow\_growth = True

sess = tf.Session(con=con)

sess.run(tf.global\_va\_initializer())

ge\_s(sess, tar\_lstm, B\_S, ged\_n, positive\_file)

gen\_d\_l.c\_bes(positive\_file)

for w in xrange(PRE\_W\_N):

l = pre\_train\_w(sess, ge, gen\_d\_l)

if w % 5 == 0:

ge\_s(sess, ge, B\_S, ged\_n, eval\_file)

like\_d\_l.c\_bes(eval\_file)

t\_l = tar\_l(sess, tar\_lstm, like\_d\_l)

for \_ in range(50):

ge\_s(sess, ge, B\_S, ged\_n, negative\_file)

dis\_d\_l.load\_train\_d(positive\_file, negative\_file)

for \_ in range(3):

dis\_d\_l.re\_po()

for it in xrange(dis\_d\_l.n\_b):

x\_b, y\_b = dis\_d\_l.n\_b()

f = {

dis.i\_x: x\_b,

dis.i\_y: y\_b,

dis.do\_keep\_prob: dis\_do\_keep\_prob

}

\_ = sess.run(dis.train\_op, f)

r = R(ge, 0.8)

for total\_b in range(TOTAL\_B):

for it in range(1):

s = ge.generate(sess)

res = r.get\_re(sess, s, 16, dis)

f = {ge.x: s, ge.res: res}

\_ = sess.run(ge.g\_updates, f\_dict=f)

if total\_b % 5 == 0 or total\_b == TOTAL\_B - 1:

ge\_s(sess, ge, B\_S, ged\_n, eval\_file)

like\_d\_l.c\_bes(eval\_file)

t\_l = tar\_l(sess, tar\_lstm, like\_d\_l)

r.update\_pa()

for \_ in range(5):

ge\_s(sess, ge, B\_S, ged\_n, negative\_file)

dis\_d\_l.load\_train\_d(positive\_file, negative\_file)

for \_ in range(3):

dis\_d\_l.re\_po()

for it in xrange(dis\_d\_l.n\_b):

x\_b, y\_b = dis\_d\_l.n\_b()

f = {

dis.i\_x: x\_b,

dis.i\_y: y\_b,

dis.do\_keep\_prob: dis\_do\_keep\_prob

}

\_ = sess.run(dis.train\_op, f)

main()