项目1 基于马尔科夫模型的自动即兴音乐推荐

1.3 模块实现

1.3.1 钢琴伴奏制作

1.和弦的实现

本部分代码完成添加和弦功能。

#midi\_extended/Track.py中的TrackExtended类

def my\_chorus\_num(self, all\_note, length, num=1, velocity=70, channel=0, change=1): #change为1表示级数调用（索引音阶），0表示和弦名称调用（无需索引音阶）

base\_note = 60 #基音

base\_num = 0

delay = 0

for chord\_note in all\_note:

for j in range(num):

count = 0

for note in chord\_note:

note = base\_note+base\_num\*12 + sum(self.scale[0:note]) if change \

else base\_note + base\_num \* 12 + note - 1

#绝对和弦名称时直接加上得到的数字，级数表示和弦需要索引音阶内的音符

super().append(Message('note\_on', note=note, velocity=velocity if count else velocity + 10, time=round(delay \* self.meta\_time), channel=channel))

for note in chord\_note:

note = base\_note + base\_num \* 12 + sum(self.scale[0:note]) if change else base\_note + base\_num \* 12 + note - 1

super().append(Message('note\_off', note=note, velocity=velocity if count else velocity + 10, time=0 if count else round(0.96 \* self.meta\_time \* length), channel=channel))

count = count + 1 #设第一排为重拍

2.和弦级数转为当前调式音阶

将用户输入的和弦级数（数字1~7）转换为绝对的和弦名称，再用musthe()方法将和弦名称转为当前调式下的音阶。

def change(self, num, key, mode, count=4):

#将和弦从数字更改为字符串

d = {'C': 1, 'D': 2, 'E': 3, 'F': 4, 'G': 5, 'A': 6, 'B': 7}

result = []

try:

if type(num) == int and 0 < num < 8:

s = musthe.Scale(musthe.Note(key), mode)

scale = []

for i in range(len(s)):

scale.append(str(s[i]))

for i in range(count):

try: #对音阶数7取余

result.append(d[scale[(num - 1 + 2 \* i) % 7][0]])

except IndexError:

#五声音阶的数目比较少，可能会存在超出索引范围的现象，对5取余

result.append(d[scale[(num - 1 + 2 \* i) % 5][0]])

else:

raise TypeError('num should be int from 1~7.')

except NameError:

return self.change(num, key, 'aeolian', count)

return result

3.根据预置节奏生成伴奏

本项目为4/4拍和3/4拍的歌曲各提供了6种预置节奏型，共12种节奏型的钢琴伴奏，其中预置节奏型由日常演奏经验而得，是常用的节奏型，能适应大多数曲目。

def my\_chorus\_4\_simple(self, chord\_progression, type=1, change=1, circulation=1): #4/4拍节奏预置

def mode1(): #模式1

if change:

for chord in chord\_progression:

self.my\_chorus(chord, 4)

else:

self.my\_chorus\_num(chord\_progression, 4)

def mode2(): #模式2

if change:

for chord in chord\_progression:

self.my\_chorus(chord, 1, 1, 80)

self.my\_chorus(chord, 1, 1)

self.my\_chorus(chord, 1, 1, 60)

self.my\_chorus(chord, 0.5, 2)

else:

l = len(chord\_progression)

for j in range(circulation):

self.my\_chorus\_num([chord\_progression[j % l]], 1, 1, 80)

self.my\_chorus\_num([chord\_progression[j % l]], 1, 1)

self.my\_chorus\_num([chord\_progression[j % l]], 1, 1, 60)

self.my\_chorus\_num([chord\_progression[j % l]], 0.5, 2)

def mode3(): #模式3

if change:

for chord in chord\_progression:

self.my\_chorus(chord, 1, 1, 80)

self.my\_chorus(chord, 1, 1)

self.my\_chorus(chord, 1.5, 1, 60)

self.my\_chorus(chord, 0.5)

else:

l = len(chord\_progression)

for j in range(circulation):

self.my\_chorus\_num([chord\_progression[j % l]], 1, 1, 80)

self.my\_chorus\_num([chord\_progression[j % l]], 1, 1)

self.my\_chorus\_num([chord\_progression[j % l]], 1.5, 1, 60)

self.my\_chorus\_num([chord\_progression[j % l]], 0.5)

def mode4(): #模式4

if change:

for chord in chord\_progression:

self.my\_chorus(chord, 2, 1, 80)

self.my\_chorus(chord, 1.5, 1, 60)

self.my\_chorus(chord, 0.5)

else:

l = len(chord\_progression)

for j in range(circulation):

self.my\_chorus\_num([chord\_progression[j % l]], 2, 1, 80)

self.my\_chorus\_num([chord\_progression[j % l]], 1.5, 1, 60)

self.my\_chorus\_num([chord\_progression[j % l]], 0.5)

def mode5(): #模式5

if change:

for chord in chord\_progression:

self.my\_chorus(chord, 0.5, 1, 80)

self.my\_chorus(chord, 0.5, 2)

self.my\_chorus(chord, 1)

self.my\_chorus(chord, 0.5, 1, 60)

self.my\_chorus(chord, 0.5, 2)

else:

l = len(chord\_progression)

for j in range(circulation):

self.my\_chorus\_num([chord\_progression[j % l]], 0.5, 1, 80)

self.my\_chorus\_num([chord\_progression[j % l]], 0.5, 2)

self.my\_chorus\_num([chord\_progression[j % l]], 1)

self.my\_chorus\_num([chord\_progression[j % l]], 0.5, 1, 60)

self.my\_chorus\_num([chord\_progression[j % l]], 0.5, 2)

def mode6(): #模式6

if change:

for chord in chord\_progression:

self.my\_chorus(chord, 0.5, 1, 80)

self.my\_chorus(chord, 0.5, 2)

self.my\_chorus(chord, 0.25, 4)

self.my\_chorus(chord, 0.5, 1, 60)

self.my\_chorus(chord, 0.5, 1)

self.my\_chorus(chord, 0.25, 2)

else:

l = len(chord\_progression)

for j in range(circulation):

self.my\_chorus\_num([chord\_progression[j % l]], 0.5, 1, 80)

self.my\_chorus\_num([chord\_progression[j % l]], 0.5, 2)

self.my\_chorus\_num([chord\_progression[j % l]], 0.25, 4)

self.my\_chorus\_num([chord\_progression[j % l]], 0.5, 1, 60)

self.my\_chorus\_num([chord\_progression[j % l]], 0.5, 1)

self.my\_chorus\_num([chord\_progression[j % l]], 0.25, 2)

type\_d = {1: mode1, 2: mode2, 3: mode3, 4: mode4, 5: mode5, 6: mode6}

type\_d.get(type)()

def my\_chorus\_3\_simple(self, chord\_progression, type=1, change=1, circulation=1): #3/4拍的预置

def mode1(): #模式1

if change:

for chord in chord\_progression:

self.my\_chorus(chord, 3)

else:

self.my\_chorus\_num(chord\_progression, 3)

def mode2(): #模式2

if change:

for chord in chord\_progression:

self.my\_chorus(chord, 1, 1, 80)

self.my\_chorus(chord, 1, 2)

else:

l = len(chord\_progression)

for j in range(circulation):

self.my\_chorus\_num([chord\_progression[j % l]], 1, 1, 80)

self.my\_chorus\_num([chord\_progression[j % l]], 1, 2)

def mode3(): #模式3

if change:

for chord in chord\_progression:

self.my\_chorus(chord, 1, 1, 80)

self.my\_chorus(chord, 0.5, 4)

else:

l = len(chord\_progression)

for j in range(circulation):

self.my\_chorus\_num([chord\_progression[j % l]], 1, 1, 80)

self.my\_chorus\_num([chord\_progression[j % l]], 0.5, 4)

def mode4(): #模式4

if change:

for chord in chord\_progression:

self.my\_chorus(chord, 0.5, 1, 80)

self.my\_chorus(chord, 1)

self.my\_chorus(chord, 0.5, 3)

else:

l = len(chord\_progression)

for j in range(circulation):

self.my\_chorus\_num([chord\_progression[j % l]], 0.5, 1, 80)

self.my\_chorus\_num([chord\_progression[j % l]], 1)

self.my\_chorus\_num([chord\_progression[j % l]], 0.5, 3)

def mode5(): #模式5

if change:

for chord in chord\_progression:

self.my\_chorus(chord, 1.5, 1, 80)

self.my\_chorus(chord, 0.5, 3)

else:

l = len(chord\_progression)

for j in range(circulation):

self.my\_chorus\_num([chord\_progression[j % l]], 1.5, 1, 80)

self.my\_chorus\_num([chord\_progression[j % l]], 0.5, 3)

def mode6(): #模式6

if change:

for chord in chord\_progression:

self.my\_chorus(chord, 0.75, 1, 80)

self.my\_chorus(chord, 0.25)

self.my\_chorus(chord, 0.75)

self.my\_chorus(chord, 0.25)

self.my\_chorus(chord, 0.75)

self.my\_chorus(chord, 0.25)

else:

l = len(chord\_progression)

for j in range(circulation):

self.my\_chorus\_num([chord\_progression[j % l]], 0.75, 1, 80)

self.my\_chorus\_num([chord\_progression[j % l]], 0.25)

self.my\_chorus\_num([chord\_progression[j % l]], 0.75)

self.my\_chorus\_num([chord\_progression[j % l]], 0.25)

self.my\_chorus\_num([chord\_progression[j % l]], 0.75)

self.my\_chorus\_num([chord\_progression[j % l]], 0.25)

type\_d = {1: mode1, 2: mode2, 3: mode3, 4: mode4, 5: mode5, 6: mode6}

type\_d.get(type)()

1.3.2 乐句生成

1.添加音符

#向音轨内添加一个普通音符，此函数由MusicCritique提供

def add\_note(self, note, length, modulation=0, base\_num=0, delay=0, velocity=90, scale=[0, 2, 2, 1, 2, 2, 2, 1], channel=0, pitch\_type=0, tremble\_setting=None, bend\_setting=None):

bpm = self.bpm

base\_note = 60 + modulation

if pitch\_type == 0:

try:

super().append(Message('note\_on', note=base\_note + base\_num \* 12 + sum(scale[0:note]), velocity=velocity, time=round(delay \* self.meta\_time), channel=channel))

super().append(Message('note\_off', note=base\_note + base\_num \* 12 + sum(scale[0:note]), velocity=velocity, time=int(round(0.96 \* self.meta\_time \* length)), channel=channel))

except IndexError: #选中五声音阶时，只有五个音，而hmm最多生成七个音

super().append(Message('note\_on', note=base\_note + base\_num \* 12 + sum(scale[0:note - 2]), velocity=velocity, time=round(delay \* self.meta\_time), channel=channel))

super().append(Message('note\_off', note=base\_note + base\_num \* 12 + sum(scale[0:note - 2]), velocity=velocity, time=int(round(0.96 \* self.meta\_time \* length)), channel=channel))

elif pitch\_type == 1: #颤音

try:

pitch = tremble\_setting['pitch']

wheel\_times = tremble\_setting['wheel\_times']

super().append(Message('note\_on', note=base\_note + base\_num \* 12 + sum(scale[0:note]), velocity=velocity, time=round(delay \* self.meta\_time), channel=channel))

for i in range(wheel\_times):

super().append(Message('pitchwheel', pitch=pitch, time=round(0.96 \* self.meta\_time \* length / (2 \* wheel\_times)), channel=channel))

super().append(Message('pitchwheel', pitch=0, time=0, channel=channel))

super().append(Message('pitchwheel', pitch=-pitch, time=round(0.96 \* self.meta\_time \* length / (2 \* wheel\_times)), channel=channel))

super().append(Message('pitchwheel', pitch=0, time=0, channel=channel))

super().append(Message('note\_off', note=base\_note + base\_num \* 12 + sum(scale[0:note]), velocity=velocity, time=0, channel=channel))

except:

print(traceback.format\_exc())

elif pitch\_type == 2:

try:

pitch = bend\_setting['pitch']

PASDA = bend\_setting['PASDA']

#结合PASDA(Prepare-Attack-Sustain-Decay-Aftermath)属性值实现MIDI滑音和颤音效果

prepare\_rate = PASDA[0] / sum(PASDA)

attack\_rate = PASDA[1] / sum(PASDA)

sustain\_rate = PASDA[2] / sum(PASDA)

decay\_rate = PASDA[3] / sum(PASDA)

aftermath\_rate = PASDA[4] / sum(PASDA)

super().append(Message('note\_on', note=base\_note + base\_num \* 12 + sum(scale[0:note]), velocity=round(100 \* velocity), time=round(delay \* self.meta\_time), channel=channel))

super().append(Message('aftertouch', time=round(0.96 \* self.meta\_time \* length \* prepare\_rate), channel=channel))

super().append(Message('pitchwheel', pitch=pitch, time=round(0.96 \* self.meta\_time \* length \* attack\_rate), channel=channel))

super().append(Message('aftertouch', time=round(0.96 \* self.meta\_time \* length \* sustain\_rate), channel=channel))

super().append(Message('pitchwheel', pitch=0, time=round(0.96 \* self.meta\_time \* length \* decay\_rate), channel=channel))

super().append(Message('note\_off', note=base\_note + base\_num \* 12 + sum(scale[0:note]), velocity=velocity, time=round(0.96 \* self.meta\_time \* length \* aftermath\_rate), channel=channel))

except:

print(traceback.format\_exc())

def add\_rest(self, length, velocity=80, channel=0): #增加休止符

super().append(Message('note\_off', note=0, velocity=velocity, time=round(0.96 \* self.meta\_time \* length),

channel=channel))

def add\_tenuto(self, length): #增加延音音符

off = super().pop() #list的最后一个音符note\_off

on = super().pop() #list的最后一个音符note\_on

off.time = round(off.time + 0.96 \* self.meta\_time \* length)

super().append(on)

super().append(off)

2.旋律生成

相关代码如下：

def hmmmelody():

startprob = np.array([0.15, 0.15, 0.15, 0.15, 0.15, 0.15, 0.10, 0.00, 0.00]) #初始分布

#状态转移矩阵，由日常演奏经验得出

transmat = np.array([[0.05, 0.10, 0.20, 0.15, 0.20, 0.10, 0.05, 0.05, 0.10], [0.10, 0.05, 0.10, 0.20, 0.20, 0.10, 0.10, 0.05, 0.10], [0.20, 0.10, 0.05, 0.10, 0.10, 0.20, 0.10, 0.05, 0.10], [0.10, 0.10, 0.20, 0.05, 0.10, 0.10, 0.20, 0.05, 0.10], [0.10, 0.20, 0.10, 0.10, 0.05, 0.10, 0.20, 0.05, 0.10], [0.05, 0.10, 0.20, 0.25, 0.10, 0.05, 0.10, 0.05, 0.10], [0.05, 0.10, 0.20, 0.10, 0.25, 0.10, 0.05, 0.05, 0.10], [0.12, 0.12, 0.12, 0.12, 0.12, 0.12, 0.12, 0.16, 0.00], [0.12, 0.12, 0.12, 0.12, 0.12, 0.12, 0.12, 0.16, 0.00]])

means = np.array([[1], [2], [3], [4], [5], [6], [7], [0], [-1]])

#covariance为协方差

covars = .000000000001 \* np.tile(np.identity(1), (9, 1, 1))

#identity的参数1要和means每一行中列数对应

#np.identity 制造对角阵，使用np.tile把对角阵复制成4行1列1条的三维矩阵

model = hmm.GaussianHMM(n\_components=9, covariance\_type="full")

model.startprob\_ = startprob

model.transmat\_ = transmat

model.means\_ = means

model.covars\_ = covars

#产生样本

X, Z = model.sample(50)

m = []

for i in range(50):

temp = int(round(X[i, 0]))

m.append(temp)

#print(m)

return m

3.节奏生成

相关代码如下：

def hmmrhythm():

#初始概率

startprob = np.array([0.15, 0.15, 0.20, 0.20, 0.00, 0.00, 0.20, 0.10, 0.00])

transmat = np.array([[0.15, 0.15, 0.10, 0.10, 0.10, 0.10, 0.10, 0.10, 0.10], [0.20, 0.20, 0.20, 0.10, 0.05, 0.05, 0.10, 0.05, 0.05], [0.05, 0.20, 0.20, 0.20, 0.10, 0.05, 0.05, 0.10, 0.05], [0.05, 0.05, 0.20, 0.20, 0.20, 0.10, 0.05, 0.05, 0.10], [0.10, 0.05, 0.05, 0.20, 0.20, 0.20, 0.10, 0.05, 0.05], [0.05, 0.10, 0.05, 0.05, 0.20, 0.20, 0.20, 0.10, 0.05], [0.05, 0.05, 0.10, 0.05, 0.05, 0.20, 0.20, 0.20, 0.10], [0.10, 0.05, 0.05, 0.10, 0.05, 0.05, 0.20, 0.20, 0.20], [0.20, 0.10, 0.05, 0.05, 0.10, 0.05, 0.05, 0.20, 0.20]])

#每个分量的均值

means = np.array([[1], [2], [4], [8], [16], [32], [6], [12], [24]])

#每个分量的协方差

covars = .000000000001 \* np.tile(np.identity(1), (9, 1, 1))

#identity的参数1要和means每一行中的列数对应

#np.identity 制造对角阵，使用np.tile把对角阵复制成4行1列1条的三维矩阵

#建立HMM实例并设置参数

model = hmm.GaussianHMM(n\_components=9, covariance\_type="full")

model.startprob\_ = startprob

model.transmat\_ = transmat

model.means\_ = means

model.covars\_ = covars

#产生样本

X, Z = model.sample(32)

for i in range(32):

X[i, 0] = int(round(X[i, 0]))

r = X[:, 0]

sum = 0

i = 0

new\_r = []

while 1 - sum > 0:

sum += 1 / r[i]

new\_r.append(1 / r[i])

i += 1

new\_r[i - 1] = 0

new\_r[i - 1] = 1 - np.sum(new\_r)

#print(r[0:i - 1])

#print(new\_r)

#print(np.sum (new\_r))

return new\_r

1.3.3 贝斯伴奏制作

1.添加贝斯轨

向歌曲内添加贝斯轨的相关代码如下：

#向音轨内添加一个普通音符，此函数由MusicCritique提供

def add\_bass(self, note, length, base\_num=-2, velocity=1.0, channel=6, delay=0):

bpm = self.bpm

scale = self.scale

base\_note = 60

super().append(Message('note\_on', note=base\_note + base\_num \* 12 + sum(self.scale[0:note]), velocity=round(80 \* velocity), time=round(delay \* self.meta\_time), channel=channel))

super().append(Message('note\_off', note=base\_note + base\_num \* 12 + sum(self.scale[0:note]), velocity=round(80 \* velocity), time=int(round(0.96 \* self.meta\_time \* length)), channel=channel))

2.预置贝斯轨

根据日常演奏经验，对4/4拍和3/4拍的歌曲分别给出7种常用贝斯轨，共14种，其中函数的输入chord\_progression可以是和弦名称，也可以是和弦级数。例如：self.chord\_progression = ['Fmaj7', 'Em7', 'Dm7', 'Cmaj7']或self.chord\_progression = '4321'。

def my\_bass\_4\_simple(self, chord\_progression, type=1, change=1):

d = {'C': 1, 'D': 2, 'E': 3, 'F': 4, 'G': 5, 'A': 6, 'B': 7}

#这里只用7个音，因为add\_bass内部已经有对音阶的索引，所以不用12个音

def mode1():

for chord in chord\_progression:

self.add\_bass(d[chord[0]] if change else int(chord),0.25)#根音

self.add\_rest(0.25) #休止

self.add\_bass(d[chord[0]] if change else int(chord), 0.25)

self.add\_rest(0.5)

self.add\_bass(d[chord[0]] if change else int(chord), 0.25)

self.add\_rest(0.5)

self.add\_bass(d[chord[0]] if change else int(chord), 1)

self.add\_bass((d[chord[0]] + 4) % 7 if change else (int(chord) + 4) % 7, 1) #五音

#此处对和弦名与和弦级数做了兼容处理，函数的输入可以是和弦名也可以是和弦级数

#change为1表示和弦名，为0表示和弦级数，在add\_bass里会做相应的处理

def mode2():

for chord in chord\_progression:

for i in range(16): #十六音符根音

self.add\_bass(d[chord[0]] if change else int(chord), 0.25)

def mode3():

for chord in chord\_progression:

self.add\_bass(d[chord[0]] if change else int(chord),0.5)#根音

self.add\_bass(d[chord[0]] if change else int(chord), 0.5)

self.add\_rest(0.5)

self.add\_bass((d[chord[0]] + 4) % 7 if change else (int(chord) + 4) % 7, 0.5) #五音

self.add\_bass(d[chord[0]] if change else int(chord), 0.5)

self.add\_bass(d[chord[0]] if change else int(chord), 0.5)

self.add\_rest(0.5)

self.add\_bass((d[chord[0]] - 1) % 7 if change else (int(chord) - 1) % 7, 0.5) #七音

def mode4():

for chord in chord\_progression: #行进贝司

self.add\_bass(d[chord[0]] if change else int(chord), 1)#根音

self.add\_bass((d[chord[0]] + 4) % 7 if change else (int(chord) + 4) % 7, 1) #五音

self.add\_bass(d[chord[0]] if change else int(chord), 1)#根音

self.add\_bass((d[chord[0]] - 1) % 7 if change else (int(chord) - 1) % 7, 0.75) #七音

self.add\_bass((d[chord[0]] - 1) % 7 if change else (int(chord) - 1) % 7, 0.25) #七音

def mode5():

for chord in chord\_progression:

self.add\_bass(d[chord[0]] if change else int(chord), 1)#根音

self.add\_bass(d[chord[0]] if change else int(chord), 1)#根音

self.add\_bass(d[chord[0]] if change else int(chord),1.5)#根音

self.add\_bass((d[chord[0]] - 1) % 7 if change else (int(chord) - 1) % 7, 0.5) #七音

def mode6():

i = 0

for chord in chord\_progression: #击弦贝司

if i % 2 == 0:

self.add\_bass(d[chord[0]] if change else int(chord), 0.25, channel=8) #根音

self.add\_rest(0.5)

self.add\_bass((d[chord[0]] + 4) % 7 if change else (int(chord) + 4) % 7, 0.25, channel=8) #五音

self.add\_rest(0.5)

self.add\_bass(d[chord[0]] if change else int(chord), 0.5, channel=8) #根音

self.add\_bass(d[chord[0]] if change else int(chord), 0.25, channel=8) #根音

self.add\_rest(0.5)

self.add\_bass((d[chord[0]] + 4) % 7 if change else (int(chord) + 4) % 7, 0.25, channel=8) #五音

self.add\_rest(0.5)

self.add\_bass(d[chord[0]] if change else int(chord), 0.5, channel=8) #根音

else:

self.add\_bass(d[chord[0]] if change else int(chord), 0.25, channel=8) #根音

self.add\_rest(0.5)

self.add\_bass(d[chord[0]] if change else (int(chord) + 4) % 7, 0.25, channel=8) #五音

self.add\_rest(0.5)

self.add\_bass(d[chord[0]] if change else int(chord), 0.5, channel=8) #根音

self.add\_bass(d[chord[0]] if change else int(chord), 0.5, channel=8) #根音

self.add\_bass(d[chord[0]] if change else int(chord), 0.5, channel=8) #根音

self.add\_bass((d[chord[0]] + 4) % 7 if change else (int(chord) + 4) % 7, 0.5, channel=8) #五音

self.add\_bass(d[chord[0]] if change else int(chord), 0.5, channel=8) #根音

i += 1

def mode7():

i = 0

for chord in chord\_progression:

i += 1

if i % 2:

elf.add\_bass(d[chord[0]] if change else int(chord), 2)#根音

self.add\_bass((d[chord[0]] + 4) % 7 if change else (int(chord) + 4) % 7, 2) #五音

else:

self.add\_bass(d[chord[0]] if change else int(chord), 2) #根音

self.add\_bass((d[chord[0]] + 4) % 7 if change else (int(chord) + 4) % 7, 1) #五音

self.add\_bass(d[chord[0]] if change else int(chord), 1) #根音

type\_d = {1: mode1, 2: mode2, 3: mode3, 4: mode4, 5: mode5, 6: mode6, 7: mode7}

type\_d.get(type)()

def my\_bass\_3\_simple(self, chord\_progression, type=1, change=1):

d = {'C': 1, 'D': 2, 'E': 3, 'F': 4, 'G': 5, 'A': 6, 'B': 7}

def mode1():

i = 0

for chord in chord\_progression:

i += 1

if i % 2:

self.add\_bass(d[chord[0]] if change else int(chord), 2) #根音

self.add\_bass((d[chord[0]] + 4) % 7 if change else (int(chord) + 4) % 7, 1) #五音

else:

self.add\_bass(d[chord[0]] if change else int(chord), 1) #根音

self.add\_bass((d[chord[0]] + 4) % 7 if change else (int(chord) + 4) % 7, 1) #五音

self.add\_bass(d[chord[0]] if change else int(chord), 1) #根音

def mode2():

for chord in chord\_progression:

self.add\_bass(d[chord[0]] if change else int(chord), 3) #根音

def mode3():

i = 0

for chord in chord\_progression:

i += 1

if i % 2:

self.add\_bass((d[chord[0]] - 1) % 7 if change else (int(chord) - 1) % 7, 1) #七音

self.add\_bass(d[chord[0]] if change else int(chord), 1) #根音

self.add\_bass((d[chord[0]] + 2) % 7 if change else (int(chord) + 2) % 7, 1) #三音

else:

self.add\_bass((d[chord[0]] - 1) % 7 if change else (int(chord) - 1) % 7, 1) #七音

self.add\_bass(d[chord[0]] if change else int(chord), 1) #根音

self.add\_bass((d[chord[0]] - 1) % 7 if change else (int(chord) - 1) % 7, 1) #七音

def mode4():

for chord in chord\_progression:

self.add\_bass(d[chord[0]] if change else int(chord), 1) #根音

self.add\_bass(d[chord[0]] if change else int(chord), 1.5) #根音

self.add\_bass((d[chord[0]] + 4) % 7 if change else int(chord) + 4, 0.5) #五音

def mode5():

i = 0

for chord in chord\_progression:

i += 1

if i % 2:

self.add\_bass(d[chord[0]] if change else int(chord), 0.5) #根音

self.add\_bass(d[chord[0]] if change else int(chord), 0.5) #根音

self.add\_bass((d[chord[0]] + 4) % 7 if change else (int(chord) + 4) % 7, 1) #五音

self.add\_bass(d[chord[0]] if change else int(chord), 1) #根音

else:

self.add\_bass(d[chord[0]] if change else int(chord), 0.5) #根音

self.add\_bass(d[chord[0]] if change else int(chord), 0.5) #根音

self.add\_bass((d[chord[0]] + 4) % 7 if change else (int(chord) + 4) % 7, 1) #五音

self.add\_bass(d[chord[0]] % 7 if change else int(chord), 0.5) #根音

self.add\_bass((d[chord[0]] - 1) % 7 if change else (int(chord) - 1) % 7, 0.5) #七音

def mode6():

for chord in chord\_progression:

self.add\_bass(d[chord[0]] if change else int(chord), 0.5) #根音

self.add\_bass(d[chord[0]] if change else int(chord), 0.5) #根音

self.add\_bass((d[chord[0]] + 4) % 7 if change else (int(chord) + 4) % 7, 0.5) #五音

self.add\_bass(d[chord[0]] if change else int(chord), 0.5) #根音

self.add\_bass((d[chord[0]] - 1) % 7 if change else (int(chord) - 1) % 7, 0.5) #五音

self.add\_bass(d[chord[0]] if change else int(chord), 0.5) #根音

def mode7():

for chord in chord\_progression:

self.add\_bass(d[chord[0]] if change else int(chord), 0.25) #根音

self.add\_rest(0.25) #根音

self.add\_bass(d[chord[0]] if change else int(chord), 0.25) #根音

self.add\_rest(0.5) #根音

self.add\_bass((d[chord[0]] + 4) % 7 if change else int(chord) + 4, 0.25) #五音

self.add\_rest(0.5) #根音

self.add\_bass(d[chord[0]] if change else int(chord), 0.5) #根音

self.add\_bass((d[chord[0]] - 1) % 7 if change else (int(chord) - 1) % 7, 0.5) #根音

type\_d = {1: mode1, 2: mode2, 3: mode3, 4: mode4, 5: mode5, 6: mode6, 7: mode7}

type\_d.get(type)()

1.3.4 汇总歌曲制作

1.日志模块

制作歌曲时记录日志，在当前目录下生成日志文件。

import logging

from functools import wraps

def decorator\_log(fun):

logging.basicConfig(level=logging.DEBUG, format='%(asctime)s %(filename)s[line:%(lineno)d] %(levelname)s %(message)s', datefmt='%a, %d %b %Y %H:%M:%S', filename='./test.log', filemode='w')

@wraps(fun)

def fun\_in(\*args, \*\*kwargs):

logging.debug("{} start.".format(fun.\_\_name\_\_))

fun(\*args, \*\*kwargs)

logging.debug("{} end.".format(fun.\_\_name\_\_))

return fun\_in

2.音乐可视化

相关代码如下：

#将音乐的旋律高低与时值长短可视化，保存图片于my/data下，函数由MusicCritique提供

def piano\_roll\_test(self):

path = self.file\_path

mid = MidiFileExtended(path, 'r')

mid.turn\_track\_into\_numpy\_matrix('Piano', "../my/data/Piano.npy")

mid.generate\_track\_from\_numpy\_matrix("../my/data/Piano.npy", (288, 128), 'Piano', False, True, '../my/data/Piano.png')

mid.turn\_track\_into\_numpy\_matrix('Melody', "../my/data/Melody.npy")

mid.generate\_track\_from\_numpy\_matrix("../my/data/Melody.npy", (288, 128), 'Melody', False, True, '../my/data/Melody.png')

if self.sw\_bass:

mid.turn\_track\_into\_numpy\_matrix('Bass', "../my/data/Bass.npy")

mid.generate\_track\_from\_numpy\_matrix("../my/data/Bass.npy",(288, 128), 'Bass', False, True, '../my/data/Bass.png')

3.Impromptu类

后台所有功能汇总类，在此实现即兴曲目的信息输入，可以保存并播放文件，实现音乐可视化。

print("Import start.") #初次运行import时间较长

from midi\_extended.MidiFileExtended import MidiFileExtended

import my.Q\_myhmm

import my.decorator\_log

import numpy as np

import time

print("Import end.")

class Impromptu:

"""

根据和弦进程自动生成旋律。

bpm：正整数，每分钟心跳数

time\_signature：每个小节中的节拍数，通常为n/m

默认的节奏伴奏目前仅支持4/4或3/4

Key：音符可以是C D E F G A B

Mode：定即兴的规模

file\_path：MIDI文件的路径和名称

chord\_progression：和弦列表

intensity：水平即兴演奏和垂直即兴演奏的参数

repeat：重复相同和弦进行的时间

"""

def \_\_init\_\_(self):

self.bpm = 120

self.time\_signature = '4/4'

self.key = 'C'

self.mode = 'major'

self.file\_path = '../my/data/song.mid'

#self.chord\_progression = ['Fmaj7', 'Em7', 'Dm7', 'Cmaj7']

#self.chord\_progression = ['Cmaj7', 'Am7', 'F', 'E7']

self.chord\_progression = '4321'

#列表是已经整理好数据类型，即列表内全是字符（和弦名）或全是级数（由GUI 检查）

self.intensity = 0

self.repeat = 1

self.mid = MidiFileExtended(self.file\_path, type=1, mode='w')

self.accompany\_type = 1

self.sw\_bass = False

self.bass\_type = 1

self.silent = False

self.accompany\_tone = 4 if self.silent else 0

self.note\_tone = 26 if self.silent else 0

@property

def scale(self): #定义各种参数

d = {'major': [0, 2, 2, 1, 2, 2, 2, 1],

'dorian': [0, 2, 1, 2, 2, 2, 1, 2],

'phrygian': [0, 1, 2, 2, 2, 1, 2, 2],

'lydian': [0, 2, 2, 2, 1, 2, 2, 1],

'mixolydian': [0, 2, 2, 1, 2, 2, 1, 2],

'minor': [0, 2, 1, 2, 2, 1, 2, 2],

'locrian': [0, 1, 2, 2, 1, 2, 2, 2],

'major\_pentatonic': [0, 2, 2, 3, 2, 3],

'minor\_pentatonic': [0, 3, 2, 2, 3, 2]}

aliases = {'Ionian': 'major', 'aeolian': 'minor'}

try:

self.\_scale = d[self.mode.lower()]

except KeyError: #异常处理

try:

self.\_scale = d[aliases[self.mode]]

except KeyError:

raise KeyError('Can not find your mode. Please check your key.')

return self.\_scale

@my.decorator\_log.decorator\_log

def chorus(self):

track = self.mid.get\_extended\_track('Piano')

track.scale = self.scale

if type(self.chord\_progression) == list: #绝对的和弦名称

for i in range(self.repeat):

if self.time\_signature[0] == '4': track.my\_chorus\_4\_simple(chord\_progression=self.chord\_progression, type=2)

else: track.my\_chorus\_3\_simple(chord\_progression=self.chord\_progression, type=2)

else: #是级数表示的和弦

change\_result = []

for chord in self.chord\_progression:

change\_result.append(track.change(int(chord),self.key, self.mode))

for i in range(self.repeat):

if self.time\_signature[0] == '4':

track.my\_chorus\_4\_simple(change\_result, type=1, change=0, circulation=len(self.chord\_progression))

else:

track.my\_chorus\_3\_simple(change\_result, type=1, change=0, circulation=len(self.chord\_progression))

print("To specify the accompaniment, you can also call function in ./midi\_extended/Track.py/my\_chorus")

help(track.my\_chorus)

@my.decorator\_log.decorator\_log

def note(self):

track = self.mid.get\_extended\_track('Melody')

#track.print\_msgs()

for j in range(self.repeat):

print("\r note {} is making...".format(j + 1), end="")

for chord in self.chord\_progression:

melody = my.Q\_myhmm.hmmmelody()

rhythm = my.Q\_myhmm.hmmrhythm()

multiple = int(self.time\_signature[0])

d = {'C': 1, 'Db': 2, 'D': 3, 'Eb': 4, 'E': 5, 'F': 6, 'Gb': 7, 'G': 8, 'Ab': 9, 'A': 10, 'Bb': 11,

'B': 12, 'C#': 2, 'D#': 4, 'F#': 7, 'G#': 9, 'A#': 11}

if self.intensity:

np.random.seed(round(1000000 \* time.time()) % 100) # Seed must be between 0 and 2\*\*32 - 1

p = np.array([self.intensity, 1 - self.intensity])

if type(self.chord\_progression) == list: #已经是绝对的和弦名称

modulation = np.random.choice([0, d[chord[0]]], p=p.ravel())

else: #是级数

modulation = np.random.choice([0, sum(self.scale[0:int(chord)])], p=p.ravel())

else:

modulation = 0

#modulation = sum(self.scale[0:d[chord[0]]])

for i in range(len(rhythm)):

m = melody[i]

if m > 0: #是一个普通音符

track.add\_note(m, rhythm[i] \* multiple, modulation, 1, velocity=110, scale=self.scale, channel=3)

#sum(self.scale[0:d[chord[0]]]))

#print('add\_note', m, rhythm[i]\*multiple)

elif m == 0: #是休止符

track.add\_rest(rhythm[i] \* multiple)

#print('add\_rest', m, rhythm[i]\*multiple)

else: #是延音符

track.add\_tenuto(rhythm[i] \* multiple)

#print('add\_tenuto', m, rhythm[i]\*multiple)

def bass(self):

track = self.mid.get\_extended\_track('Bass')

track.scale = self.scale

if type(self.chord\_progression) == list: #已经是绝对的和弦名称

for i in range(self.repeat):

if self.time\_signature[0] == '4':

track.my\_bass\_4\_simple(chord\_progression=self.chord\_progression, type=1)

else: track.my\_bass\_3\_simple(chord\_progression=self.chord\_progression, type=1)

else: #是级数表示的和弦

for i in range(self.repeat):

if self.time\_signature[0] == '4':

track.my\_bass\_4\_simple(self.chord\_progression, type=i + 1, change=0)

else:

track.my\_bass\_3\_simple(self.chord\_progression, type=i + 1, change=0)

def piano\_roll\_test(self): #定义测试

path = self.file\_path #文件路径

mid = MidiFileExtended(path, 'r')

mid.turn\_track\_into\_numpy\_matrix('Piano', "../my/data/Piano.npy")

mid.generate\_track\_from\_numpy\_matrix("../my/data/Piano.npy", (288, 128), 'Piano', False, True,

'../my/data/Piano.png')

mid.turn\_track\_into\_numpy\_matrix('Melody', "../my/data/Melody.npy") #数据写入矩阵

mid.generate\_track\_from\_numpy\_matrix("../my/data/Melody.npy", (288, 128), 'Melody', False, True, '../my/data/Melody.png')

if self.sw\_bass:

mid.turn\_track\_into\_numpy\_matrix('Bass', "../my/data/Bass.npy")

mid.generate\_track\_from\_numpy\_matrix("../my/data/Bass.npy", (288, 128), 'Bass', False, True, '../my/data/Bass.png')

def write\_song(self): #定义写入歌曲

del self.mid

self.mid = MidiFileExtended(self.file\_path, type=1, mode='w')

self.mid.add\_new\_track('Piano', self.time\_signature, self.bpm, self.key, {'0': 4 if self.silent else 0}) #4})

#这里的轨道0和1音色是30，代表具体乐器音色

self.chorus()

self.mid.add\_new\_track('Melody', self.time\_signature, self.bpm, self.key, {'3': 26 if self.silent else 0}) #26

self.note()

if self.sw\_bass:

self.mid.add\_new\_track('Bass', self.time\_signature, self.bpm, self.key, {'6': 39 if self.silent else 33,

'7': 35, '8': 36})

self.bass()

if \_\_name\_\_ == '\_\_main\_\_': #主函数

silence = Impromptu()

print(silence.scale)

silence.write\_song()

silence.mid.save\_midi()

silence.piano\_roll\_test()

print("Done. Start to play.")

#silence.mid.play\_it()

1.3.5 GUI设计

1.用户界面空间初始化

此代码由制作的.ui文件经PyQt-tools转换而成，在Windows下运行。

from PyQt5 import QtCore, QtWidgets

class Ui\_MainWindow(object):

def setupUi(self, MainWindow): #设置界面

MainWindow.setObjectName("MainWindow")

MainWindow.resize(800, 680)

self.centralwidget = QtWidgets.QWidget(MainWindow)

self.centralwidget.setObjectName("centralwidget")

self.verticalLayoutWidget = QtWidgets.QWidget(self.centralwidget)

self.verticalLayoutWidget.setGeometry(QtCore.QRect(0, 0, 800, 680))

self.verticalLayoutWidget.setObjectName("verticalLayoutWidget") self.verticalLayout\_2=QtWidgets.QVBoxLayout(self.verticalLayoutWidget)

self.verticalLayout\_2.setContentsMargins(0, 0, 0, 0)

self.verticalLayout\_2.setObjectName("verticalLayout\_2")

self.verticalLayout\_3 = QtWidgets.QVBoxLayout()

self.verticalLayout\_3.setObjectName("verticalLayout\_3")

self.frame = QtWidgets.QFrame(self.verticalLayoutWidget)

self.frame.setEnabled(True)

self.frame.setStyleSheet("background-image:url(./background.jpeg);")

self.frame.setFrameShape(QtWidgets.QFrame.StyledPanel)

self.frame.setFrameShadow(QtWidgets.QFrame.Raised)

self.frame.setObjectName("frame")

self.w\_time\_signalture = QtWidgets.QComboBox(self.frame)

self.w\_time\_signalture.setGeometry(QtCore.QRect(360,220, 111, 21))

self.w\_time\_signalture.setObjectName("w\_time\_signalture")

self.w\_time\_signalture.addItem("")

self.w\_time\_signalture.addItem("")

self.w\_mode = QtWidgets.QComboBox(self.frame)

self.w\_mode.setGeometry(QtCore.QRect(360, 140, 111, 21))

self.w\_mode.setObjectName("w\_mode")

self.w\_mode.addItem("")

self.w\_mode.addItem("")

self.w\_mode.addItem("")

self.w\_mode.addItem("")

self.w\_mode.addItem("")

self.w\_mode.addItem("")

self.w\_mode.addItem("")

self.w\_mode.addItem("")

self.w\_mode.addItem("")

self.w\_bass = QtWidgets.QComboBox(self.frame)

self.w\_bass.setGeometry(QtCore.QRect(360, 300, 111, 21))

self.w\_bass.setObjectName("w\_bass")

self.w\_bass.addItem("")

self.w\_bass.addItem("")

self.w\_bass.addItem("")

self.w\_bass.addItem("")

self.w\_bass.addItem("")

self.w\_bass.addItem("")

self.w\_bass.addItem("")

self.w\_bass.addItem("")

self.w\_accompany = QtWidgets.QComboBox(self.frame)

self.w\_accompany.setGeometry(QtCore.QRect(360, 260, 111, 21))

self.w\_accompany.setObjectName("w\_accompany")

self.w\_accompany.addItem("")

self.w\_accompany.addItem("")

self.w\_accompany.addItem("")

self.w\_accompany.addItem("")

self.w\_accompany.addItem("")

self.w\_accompany.addItem("")

self.w\_key = QtWidgets.QComboBox(self.frame)

self.w\_key.setEnabled(True)

self.w\_key.setGeometry(QtCore.QRect(360, 100, 111, 21))

self.w\_key.setMaxVisibleItems(7)

self.w\_key.setObjectName("w\_key")

self.w\_key.addItem("")

self.w\_key.addItem("")

self.w\_key.addItem("")

self.w\_key.addItem("")

self.w\_key.addItem("")

self.w\_key.addItem("")

self.w\_key.addItem("")

self.verticalLayout\_3.addWidget(self.frame)

self.verticalLayout\_2.addLayout(self.verticalLayout\_3)

self.label = QtWidgets.QLabel(self.centralwidget)

self.label.setGeometry(QtCore.QRect(190, 340, 141, 31))

self.label.setObjectName("label")

self.w\_play = QtWidgets.QPushButton(self.centralwidget)

self.w\_play.setGeometry(QtCore.QRect(360, 490, 93, 28))

self.w\_play.setAutoDefault(False)

self.w\_play.setDefault(False)

self.w\_play.setFlat(False)

self.w\_play.setObjectName("w\_play")

self.label\_7 = QtWidgets.QLabel(self.centralwidget)

self.label\_7.setGeometry(QtCore.QRect(190, 380, 121, 31))

self.label\_7.setObjectName("label\_7")

self.w\_intensity = QtWidgets.QSlider(self.centralwidget)

self.w\_intensity.setGeometry(QtCore.QRect(340, 430, 160, 22))

self.w\_intensity.setMaximum(100)

self.w\_intensity.setSingleStep(1)

self.w\_intensity.setOrientation(QtCore.Qt.Horizontal)

self.w\_intensity.setObjectName("w\_intensity")

self.w\_repeat = QtWidgets.QLineEdit(self.centralwidget)

self.w\_repeat.setGeometry(QtCore.QRect(360, 380, 113, 21))

self.w\_repeat.setInputMask("")

self.w\_repeat.setMaxLength(32767)

self.w\_repeat.setObjectName("w\_repeat")

self.w\_bpm = QtWidgets.QLineEdit(self.centralwidget)

self.w\_bpm.setGeometry(QtCore.QRect(360, 180, 113, 21))

self.w\_bpm.setObjectName("w\_bpm")

self.w\_chord\_progression = QtWidgets.QLineEdit(self.centralwidget)

self.w\_chord\_progression.setGeometry(QtCore.QRect(360,340,113,21))

self.w\_chord\_progression.setObjectName("w\_chord\_progression")

self.label\_6 = QtWidgets.QLabel(self.centralwidget)

self.label\_6.setGeometry(QtCore.QRect(190, 180, 121, 31))

self.label\_6.setObjectName("label\_6")

self.label\_5 = QtWidgets.QLabel(self.centralwidget)

self.label\_5.setGeometry(QtCore.QRect(190, 100, 121, 31))

self.label\_5.setObjectName("label\_5")

self.label\_3 = QtWidgets.QLabel(self.centralwidget)

self.label\_3.setGeometry(QtCore.QRect(190, 420, 121, 31))

self.label\_3.setObjectName("label\_3")

self.label\_2 = QtWidgets.QLabel(self.centralwidget)

self.label\_2.setGeometry(QtCore.QRect(190, 220, 121, 31))

self.label\_2.setObjectName("label\_2")

self.label\_4 = QtWidgets.QLabel(self.centralwidget)

self.label\_4.setGeometry(QtCore.QRect(190, 140, 121, 31))

self.label\_4.setObjectName("label\_4")

self.label\_8 = QtWidgets.QLabel(self.centralwidget)

self.label\_8.setGeometry(QtCore.QRect(190, 260, 121, 31))

self.label\_8.setObjectName("label\_8")

self.label\_9 = QtWidgets.QLabel(self.centralwidget)

self.label\_9.setGeometry(QtCore.QRect(190, 300, 121, 31))

self.label\_9.setObjectName("label\_9")

self.checkBox = QtWidgets.QCheckBox(self.centralwidget)

self.checkBox.setGeometry(QtCore.QRect(630, 430, 91, 19))

self.checkBox.setObjectName("checkBox")

MainWindow.setCentralWidget(self.centralwidget)

self.menubar = QtWidgets.QMenuBar(MainWindow)

self.menubar.setGeometry(QtCore.QRect(0, 0, 798, 26))

self.menubar.setObjectName("menubar")

self.menusetting = QtWidgets.QMenu(self.menubar)

self.menusetting.setObjectName("menusetting")

self.menuhelp = QtWidgets.QMenu(self.menubar)

self.menuhelp.setObjectName("menuhelp")

self.menuAbout = QtWidgets.QMenu(self.menubar)

self.menuAbout.setObjectName("menuAbout")

MainWindow.setMenuBar(self.menubar)

self.statusbar = QtWidgets.QStatusBar(MainWindow)

self.statusbar.setObjectName("statusbar")

MainWindow.setStatusBar(self.statusbar)

self.actionsetting = QtWidgets.QAction(MainWindow)

self.actionsetting.setObjectName("actionsetting")

self.actionexit = QtWidgets.QAction(MainWindow)

self.actionexit.setObjectName("actionexit")

self.actiondocument\_2 = QtWidgets.QAction(MainWindow)

self.actiondocument\_2.setObjectName("actiondocument\_2")

self.actionabout = QtWidgets.QAction(MainWindow)

self.actionabout.setObjectName("actionabout")

self.menusetting.addSeparator()

self.menusetting.addAction(self.actionsetting)

self.menusetting.addSeparator()

self.menusetting.addAction(self.actionexit)

self.menuhelp.addAction(self.actiondocument\_2)

self.menuAbout.addAction(self.actionabout)

self.menubar.addAction(self.menusetting.menuAction())

self.menubar.addAction(self.menuhelp.menuAction())

self.menubar.addAction(self.menuAbout.menuAction())

self.retranslateUi(MainWindow)

self.w\_key.setCurrentIndex(0)

QtCore.QMetaObject.connectSlotsByName(MainWindow)

def retranslateUi(self, MainWindow): #重新加载界面

\_translate = QtCore.QCoreApplication.translate

MainWindow.setWindowTitle(\_translate("MainWindow", "MainWindow"))

self.w\_time\_signalture.setItemText(0,\_translate("MainWindow","4/4"))

self.w\_time\_signalture.setItemText(1, \_translate("MainWindow","3/4"))

self.w\_mode.setItemText(0, \_translate("MainWindow", "major"))

self.w\_mode.setItemText(1, \_translate("MainWindow", "dorian"))

self.w\_mode.setItemText(2, \_translate("MainWindow", "phrygian"))

self.w\_mode.setItemText(3, \_translate("MainWindow", "lydian"))

self.w\_mode.setItemText(4, \_translate("MainWindow", "mixolydian"))

self.w\_mode.setItemText(5, \_translate("MainWindow", "minor"))

self.w\_mode.setItemText(6, \_translate("MainWindow", "locrian"))

self.w\_mode.setItemText(7,\_translate("MainWindow","major pentatonic"))

self.w\_mode.setItemText(8,\_translate("MainWindow","minor pentatonic"))

self.w\_bass.setItemText(0, \_translate("MainWindow", "None"))

self.w\_bass.setItemText(1, \_translate("MainWindow", "1"))

self.w\_bass.setItemText(2, \_translate("MainWindow", "2"))

self.w\_bass.setItemText(3, \_translate("MainWindow", "3"))

self.w\_bass.setItemText(4, \_translate("MainWindow", "4"))

self.w\_bass.setItemText(5, \_translate("MainWindow", "5"))

self.w\_bass.setItemText(6, \_translate("MainWindow", "6"))

self.w\_bass.setItemText(7, \_translate("MainWindow", "7"))

self.w\_accompany.setItemText(0, \_translate("MainWindow", "1"))

self.w\_accompany.setItemText(1, \_translate("MainWindow", "2"))

self.w\_accompany.setItemText(2, \_translate("MainWindow", "3"))

self.w\_accompany.setItemText(3, \_translate("MainWindow", "4"))

self.w\_accompany.setItemText(4, \_translate("MainWindow", "5"))

self.w\_accompany.setItemText(5, \_translate("MainWindow", "6"))

self.w\_key.setCurrentText(\_translate("MainWindow", "C"))

self.w\_key.setItemText(0, \_translate("MainWindow", "C"))

self.w\_key.setItemText(1, \_translate("MainWindow", "D"))

self.w\_key.setItemText(2, \_translate("MainWindow", "E"))

self.w\_key.setItemText(3, \_translate("MainWindow", "F"))

self.w\_key.setItemText(4, \_translate("MainWindow", "G"))

self.w\_key.setItemText(5, \_translate("MainWindow", "A"))

self.w\_key.setItemText(6, \_translate("MainWindow", "B"))

self.label.setText(\_translate("MainWindow", "Chord progression"))

self.w\_play.setText(\_translate("MainWindow", "GO"))

self.label\_7.setText(\_translate("MainWindow", "repeat"))

self.w\_repeat.setText(\_translate("MainWindow", "1"))

self.w\_bpm.setText(\_translate("MainWindow", "120"))

self.w\_chord\_progression.setText(\_translate("MainWindow","4321"))

self.label\_6.setText(\_translate("MainWindow", "bpm"))

self.label\_5.setText(\_translate("MainWindow", "key"))

self.label\_3.setText(\_translate("MainWindow", "intensity"))

self.label\_2.setText(\_translate("MainWindow", "Time signalture"))

self.label\_4.setText(\_translate("MainWindow", "mode"))

self.label\_8.setText(\_translate("MainWindow", "accompany"))

self.label\_9.setText(\_translate("MainWindow", "bass"))

self.checkBox.setText(\_translate("MainWindow", "silent"))

self.menusetting.setTitle(\_translate("MainWindow", "Menu"))

self.menuhelp.setTitle(\_translate("MainWindow", "Help"))

self.menuAbout.setTitle(\_translate("MainWindow", "About"))

self.actionsetting.setText(\_translate("MainWindow", "setting"))

self.actionexit.setText(\_translate("MainWindow", "exit"))

self.actiondocument\_2.setText(\_translate("MainWindow", "document"))

self.actionabout.setText(\_translate("MainWindow", "about"))

2.将控件绑定功能（信号与槽的绑定）

#完成基础页面布局后，对事件设定触发，并定义触发的函数

import sys

from PyQt5.QtWidgets import QApplication, QMainWindow, QMessageBox

from PyQt5.QtCore import QCoreApplication

from my import v4

#import \_thread

#import threading

import all1

play = False

class MainCode(all1.Impromptu, QMainWindow, v4.Ui\_MainWindow):

def \_\_init\_\_(self): #初始化

super().\_\_init\_\_()

QMainWindow.\_\_init\_\_(self)

v4.Ui\_MainWindow.\_\_init\_\_(self)

#super().setupUi(self)

self.setupUi(self)

self.w\_play.clicked.connect(self.go)

self.w\_key.activated[str].connect(self.set\_key)

self.w\_mode.activated[str].connect(self.set\_mode)

self.w\_bass.activated[str].connect(self.set\_bass)

self.w\_accompany.activated[str].connect(self.set\_accompany)

self.w\_time\_signalture.activated[str].connect(self.set\_time\_signalture)

self.actionexit.triggered.connect(QCoreApplication.instance().quit)

self.actiondocument\_2.triggered.connect(self.document)

self.actionsetting.triggered.connect(self.setting)

self.actionabout.triggered.connect(self.about)

def document(self): #定义文本

text = "this is ducoment"

QMessageBox.information(self, "Message", text, QMessageBox.Ok)

def setting(self): #定义设置

text = "this is setting"

QMessageBox.information(self, "Message", text, QMessageBox.Ok)

def about(self): #定义关于

text = "author: @dongmie1999\n2020.4"

QMessageBox.information(self, "Message", text, QMessageBox.Ok)

def set\_key(self, text): #定义设置键

self.key = text

def set\_mode(self, text): #定义设置模式

self.mode = text

def set\_bass(self, text): #定义设置贝斯

if text == 'None':

self.sw\_bass = False

else:

self.sw\_bass = True

self.bass\_type = int(text)

def set\_accompany(self, text): #定义设置伴奏

self.accompany\_type = int(text)

def set\_time\_signalture(self, text): #定义设置时间

self.time\_signalture = text

def go(self): #定义行进

try:

self.bpm = int(self.w\_bpm.text())

except ValueError:

text = "bpm should be an positive integer.\nrecommend: 70~150"

QMessageBox.information(self, "Message", text, QMessageBox.Ok)

return

try:

if not 0 < int(self.w\_repeat.text()) <20:

raise ValueError

except ValueError:

text = "repeat should be an positive integer.\nrecommend: 1~5"

QMessageBox.information(self, "Message", text, QMessageBox.Ok)

return

self.intensity = int(self.w\_intensity.value()/100)

try: #用级数表示和弦

for t in self.w\_chord\_progression.text():

if 0 < int(t) < 8:

pass

else:

text = "Input should be a series fo numbers.\nEach number must be between 1~7.\n" + \

"Example: 4321 or 4536251 or 1645"

QMessageBox.information(self, "Message", text, QMessageBox.Ok)

return

self.chord\_progression = self.w\_chord\_progression.text()

except ValueError: #和弦名称

self.chord\_progression=self.w\_chord\_progression.text().split(',')

#print(self.checkBox.checkState())

if self.checkBox.checkState():

self.silent = True

else:

self.silent = False

print("Song making...")

self.write\_song()

self.mid.save\_midi()

print("Done. Start to play.")

#for n in range(self.repeat):

#获取条目文本

#str\_n = 'File index{0}'.format(n)

#添加文本到列表控件中

#self.listFile.addItem(str\_n)

#实时刷新界面

#QApplication.processEvents()

#睡眠1秒

#time.sleep(1)

#thread.start\_new\_thread(self.mid.play\_it())

self.mid.play\_it()

if \_\_name\_\_ == '\_\_main\_\_': #主函数

app = QApplication(sys.argv)

md = MainCode()

md.show()

#t1 = threading.Thread(target=md.show())

#t2 = threading.Thread(target=md.go())

#t1.start()

#t1.join()

#if play:

#print("play")

#t2.start()

#t2.join()

#play = False

sys.exit(app.exec\_())

项目2 小型智能健康推荐助手

2.3 模块实现

2.3.1 疾病预测

1.数据预处理

##### 1)心脏病数据集预处理

加载数据集和数据预处理，大部分是通过Pandas库实现，相关代码如下：

#导入相应库函数

import pandas as pd

#读取心脏病数据集

df = pd.read\_csv("../Thursday9 10 11/heart.csv")

df.head()

检查数据是否有缺省值，如果有数据会显示为NaN，且当数据有缺省值时不能对数据绘图可视化。

#检查是否有缺省值

df.loc[(df['age'].isnull()) |

(df['sex'].isnull()) |

(df['cp'].isnull()) |

(df['trestbps'].isnull()) |

(df['chol'].isnull()) |

(df['fbs'].isnull()) |

(df['restecg'].isnull()) |

(df['thalach'].isnull()) |

(df['exang'].isnull()) |

(df['oldpeak'].isnull()) |

(df['slope'].isnull()) |

(df['ca'].isnull()) |

(df['target'].isnull())]

数据集没有缺省值，数据的尺度比较大，通过绘图方式观察可以检查出错误数据

#通过seaborn绘图,观察数据

sns.pairplot(df.dropna(), hue='target')

通过观察，第五列（血液中胆固醇含量）和第十行（静息血压）有部分点和其它点有较大距离，绘制数据分布图进一步分析。

#绘制血液中胆固醇数据分布

df['chol'].hist()

#绘制静息血压分布图

df[‘treatbps’].hist()

下面是改变数据类型，例如，胸痛类型，1~4是类别变量，它的大小并不具备比较性，但是训练时数值大小会影响权重。所以要把类别变量转化为伪变量，把4个类别拆成4件，分别用0，1表示有或没有。

#将类别变量转换为伪变量

a = pd.get\_dummies(df['cp'], prefix = "cp")

b = pd.get\_dummies(df['thal'], prefix = "thal")

c = pd.get\_dummies(df['slope'], prefix = "slope")

frames = [df, a, b, c]

df = pd.concat(frames, axis = 1)

#保留转换后的变量即可，删除原来的类别变量

df = df.drop(columns = ['cp', 'thal', 'slope'])

最后使用Scikit-learn的train\_test\_split自动划分训练集和测试集。

#标签是target，是否患病

y = df.target.values

x\_data = df.drop(['target'], axis = 1)

#丢弃标签，也就是最后一行target

#按4:1划分训练集测试集

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x\_data,y,test\_size = 0.2,random\_state=0)

x\_train = x\_train.T

y\_train = y\_train.T

x\_test = x\_test.T

y\_test = y\_test.T

#心脏病数据集预处理完成

##### 2)慢性肾病数据预处理

通过Pandas读取慢性肾病数据集，读取成功效果如图2-8所示。

#读取肾病数据集

df = pd.read\_csv("../Thursday9 10 11/kidney\_disease.csv")

df.head()

对数据类型进行处理，例如食欲(appet)数据为good和poor，脓细胞团(pcc)为notpresent和present，将类别变量转换为伪变量0和1。

#yes/no; abnormal/normal;present/notpresent;good/poor都转换为0/1

df[['htn','dm','cad','pe','ane']] = df[['htn','dm','cad','pe','ane']].replace(to\_replace={'yes':1,'no':0})

df[['rbc','pc']] = df[['rbc','pc']].replace(to\_replace={'abnormal':1,'normal':0})

df[['pcc','ba']] = df[['pcc','ba']].replace(to\_replace={'present':1,'notpresent':0})

df[['appet']] = df[['appet']].replace(to\_replace={'good':1,'poor':0,'no':np.nan})

df['classification'] = df['classification'].replace(to\_replace={'ckd':1.0,'ckd\t':1.0,'notckd':0.0,'no':0.0})

df.rename(columns={'classification':'class'},inplace=True)

#将对患病有积极作用的变量设为0

df['pe'] = df['pe'].replace(to\_replace='good',value=0)

df['appet'] = df['appet'].replace(to\_replace='no',value=0)

df['cad'] = df['cad'].replace(to\_replace='\tno',value=0)

df['dm'] = df['dm'].replace(to\_replace={'\tno':0,'\tyes':1,' yes':1, '':np.nan})

#ID列去掉，为了表格中数据条理清晰而建立的变量

df.drop('id',axis=1,inplace=True)

可以看出缺省值数量不小，由于数据集不大，需要采用均值归一法，对病人和正常人分别取所有测量值的平均值来填补缺省值。

#对病人所有测量值取均值

average0\_age = df.loc[df['class'] ==True, 'age'].mean()

average0\_bp = df.loc[df['class'] == True, 'bp'].mean()

average0\_sg = df.loc[df['class'] == True, 'sg'].mean()

average0\_al = df.loc[df['class'] == True, 'al'].mean()

average0\_su = df.loc[df['class'] == True, 'su'].mean()

average0\_rbc = df.loc[df['class'] == True, 'rbc'].mean()

average0\_pc = df.loc[df['class'] == True, 'pc'].mean()

average0\_pcc = df.loc[df['class'] == True, 'pcc'].mean()

average0\_ba = df.loc[df['class'] == True, 'ba'].mean()

average0\_bgr = df.loc[df['class'] == True, 'bgr'].mean()

average0\_bu = df.loc[df['class'] == True, 'bu'].mean()

average0\_sc = df.loc[df['class'] == True, 'sc'].mean()

average0\_sod = df.loc[df['class'] == True, 'sod'].mean()

average0\_pot = df.loc[df['class'] == True, 'pot'].mean()

average0\_hemo = df.loc[df['class'] == True, 'hemo'].mean()

average0\_htn = df.loc[df['class'] == True, 'htn'].mean()

average0\_dm = df.loc[df['class'] == True, 'dm'].mean()

average0\_cad = df.loc[df['class'] == True, 'cad'].mean()

average0\_appet = df.loc[df['class'] ==True, 'appet'].mean()

average0\_pe = df.loc[df['class'] == True, 'pe'].mean()

average0\_ane = df.loc[df['class'] == True, 'ane'].mean()

#对正常人所有测量值取均值

average1\_age = df.loc[df['class'] == False, 'age'].mean()

average1\_bp = df.loc[df['class'] == False, 'bp'].mean()

average1\_sg = df.loc[df['class'] == False, 'sg'].mean()

average1\_al = df.loc[df['class'] == False, 'al'].mean()

average1\_su = df.loc[df['class'] == False, 'su'].mean()

average1\_rbc = df.loc[df['class'] == False, 'rbc'].mean()

average1\_pc = df.loc[df['class'] == False, 'pc'].mean()

average1\_pcc = df.loc[df['class'] == False, 'pcc'].mean()

average1\_ba = df.loc[df['class'] == False, 'ba'].mean()

average1\_bgr = df.loc[df['class'] == False, 'bgr'].mean()

average1\_bu = df.loc[df['class'] == False, 'bu'].mean()

average1\_sc = df.loc[df['class'] == False, 'sc'].mean()

average1\_sod = df.loc[df['class'] == False, 'sod'].mean()

average1\_pot = df.loc[df['class'] == False, 'pot'].mean()

average1\_hemo = df.loc[df['class'] == False, 'hemo'].mean()

average1\_htn = df.loc[df['class'] == False, 'htn'].mean()

average1\_dm = df.loc[df['class'] == False, 'dm'].mean()

average1\_cad = df.loc[df['class'] == False, 'cad'].mean()

average1\_appet = df.loc[df['class'] == False, 'appet'].mean()

average1\_pe = df.loc[df['class'] == False, 'pe'].mean()

average1\_ane = df.loc[df['class'] == False, 'ane'].mean()

#根据是患者还是正常人，求出的均值赋给所有缺省值。如果为null，则取均值

df.loc[(df['class']==True)&(df['age'].isnull()),'age']= average0\_age

df.loc[(df['class']==True)&(df['bp'].isnull()),'bp']= average0\_bp

df.loc[(df['class']==True)&(df['sg'].isnull()),'sg']= average0\_sg

df.loc[(df['class']==True)&(df['al'].isnull()),'al']= average0\_al

df.loc[(df['class']==True)&(df['su'].isnull()),'su']= average0\_su

df.loc[(df['class']==True)&(df['rbc'].isnull()),'rbc']= average0\_rbc

df.loc[(df['class']==True)&(df['pc'].isnull()),'pc']= average0\_pc

df.loc[(df['class']==True)&(df['pcc'].isnull()),'pcc']= average0\_pcc

df.loc[(df['class']==True)&(df['ba'].isnull()),'ba']= average0\_ba

df.loc[(df['class']==True)&(df['bgr'].isnull()),'bgr']= average0\_bgr

df.loc[(df['class']==True)&(df['bu'].isnull()),'bu']= average0\_bu

df.loc[(df['class']==True) &(df['sc'].isnull()),'sc']= average0\_sc

df.loc[(df['class']==True)&(df['sod'].isnull()),'sod']= average0\_sod

df.loc[(df['class']==True)&(df['pot'].isnull()),'pot']= average0\_pot

df.loc[(df['class']==True) &(df['hemo'].isnull()),'hemo'] =average0\_hemo

df.loc[(df['class']==True) &(df['htn'].isnull()),'htn'] = average0\_htn

df.loc[(df['class']==True) &(df['dm'].isnull()),'dm'] = average0\_dm

df.loc[(df['class']==True) &(df['cad'].isnull()),'cad'] = average0\_cad

df.loc[(df['class']==True)&(df['appet'].isnull()),'appet']=average0\_appet

df.loc[(df['class']==True)&(df['pe'].isnull()),'pe'] = average0\_pe

df.loc[(df['class']==True) &(df['ane'].isnull()),'ane'] = average0\_ane

#正常人

df.loc[(df['class']==False)&(df['age'].isnull()),'age']= average1\_age

df.loc[(df['class'] ==False) &(df['bp'].isnull()),'bp'] = average1\_bp

df.loc[(df['class'] ==False) &(df['sg'].isnull()),'sg'] = average1\_sg

df.loc[(df['class'] ==False) &(df['al'].isnull()),'al'] = average1\_al

df.loc[(df['class'] ==False) &(df['su'].isnull()),'su'] = average1\_su

df.loc[(df['class'] ==False) &(df['rbc'].isnull()),'rbc'] = average1\_rbc

df.loc[(df['class'] ==False) &(df['pc'].isnull()),'pc'] = average1\_pc

df.loc[(df['class'] ==False)&(df['pcc'].isnull()),'pcc'] = average1\_pcc

df.loc[(df['class'] ==False)&(df['ba'].isnull()),'ba'] = average1\_ba

df.loc[(df['class'] ==False&(df['bgr'].isnull()),'bgr'] = average1\_bgr

df.loc[(df['class'] ==False)&(df['bu'].isnull()),'bu'] = average1\_bu

df.loc[(df['class'] ==False)&(df['sc'].isnull()),'sc'] = average1\_sc

df.loc[(df['class'] ==False)&(df['sod'].isnull()),'sod'] = average1\_sod

df.loc[(df['class'] ==False)&(df['pot'].isnull()),'pot'] = average1\_pot

df.loc[(df['class'] ==False)&(df['hemo'].isnull()),'hemo']= average1\_hemo

df.loc[(df['class'] ==False)&(df['htn'].isnull()),'htn'] = average1\_htn

df.loc[(df['class'] ==False)&(df['dm'].isnull()),'dm'] = average1\_dm

df.loc[(df['class'] ==False) &(df['cad'].isnull()),'cad'] = average1\_cad

df.loc[(df['class']==False)&(df['appet'].isnull()),'appet']=average1\_appet

df.loc[(df['class'] ==False) &(df['pe'].isnull()),'pe'] = average1\_pe

df.loc[(df['class'] ==False) &(df['ane'].isnull()),'ane'] = average1\_ane

#再次检查是否有缺省值

df.loc[(df['age'].isnull()) |

(df['bp'].isnull()) |

(df['sg'].isnull()) |

(df['al'].isnull()) |

(df['su'].isnull()) |

(df['rbc'].isnull()) |

(df['pc'].isnull()) |

(df['pcc'].isnull()) |

(df['ba'].isnull()) |

(df['bgr'].isnull()) |

(df['bu'].isnull()) |

(df['sc'].isnull()) |

(df['sod'].isnull()) |

(df['pot'].isnull()) |

(df['hemo'].isnull()) |

(df['htn'].isnull()) |

(df['dm'].isnull()) |

(df['cad'].isnull()) |

(df['appet'].isnull()) |

(df['pe'].isnull()) |

(df['ane'].isnull()) |

(df['class'].isnull())]

#使用Scikit-learn的train\_test\_split()函数自动划分训练集和测试集

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df.iloc[:,:-1], df['class'],test\_size = 0.33, random\_state=44,stratify= df['class'] )

#慢性肾病数据预处理完成

2.模型训练及保存

##### 1)定义模型结构

##### 心脏病数据集定义模型

相关代码如下：

#由于一次尝试过多参数会导致内存不足，所以分段寻找最大值

num=np.zeros(20,int)

for i in range(0,20):

num[i]=i

"""

每次将num扩大20，迭代改变随机森林中树的数量以及权重分配等参数。使用GridSearchCV自动寻找最优参数。

"""

from sklearn.ensemble import RandomForestClassifier

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

tuned\_parameters = [{'n\_estimators':num,'class\_weight':[None,{0: 0.33,1:0.67},'balanced'],'random\_state':[1]}]

rf = GridSearchCV(RandomForestClassifier(), tuned\_parameters, cv=10,scoring='f1')

rf.fit(x\_train.T, y\_train.T)

print('Best parameters:')

print(rf.best\_params\_)

#训练完成后，使用print()函数输出最佳参数，森林中有1005棵树的时候准确率最高

#保存最优参数，将最佳参数带入模型进行训练

rf\_best = rf.best\_estimator\_

rf\_best.fit(x\_train.T, y\_train.T)

acc = rf.score(x\_test.T,y\_test.T)\*100

accuracies['Random Forest'] = acc

print("Random Forest Algorithm Accuracy Score : {:.2f}%".format(acc))

#最佳参数的模型训练完成后，在测试集上计算模型准确率，达89.86%

##### (2)慢性肾病数据集定义模型

相关代码如下：

#寻找随机森林最优参数

tuned\_parameters= [{'n\_estimators':[7,8,9,10,11,12,13,14,15,16],'max\_depth':[2,3,4,5,6,None],'class\_weight':[None,{0: 0.33,1:0.67},'balanced'],'random\_state':[42]}]

clf = GridSearchCV(RandomForestClassifier(), tuned\_parameters, cv=10,scoring='f1')

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

print('Best parameters:')

print(clf.best\_params\_)

#训练完成后，使用print()函数输出最佳参数，随机森林中有7棵树时准确率最高

#将最优参数带入模型进行训练

accuracies = {}

rf = RandomForestClassifier(class\_weight=None, max\_depth= 6,n\_estimators = 7, random\_state = 42)

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

acc = rf.score(X\_test, y\_test)\*100

accuracies['Random Forest'] = acc

print("Random Forest Algorithm Accuracy Score : {:.2f}%".format(acc))

#最佳参数的模型训练完成后，在测试集上计算模型准确率，达100%

##### 2)保存模型

##### (1)心脏病模型保存

相关代码如下：

import pickle

with open("model.pkl", "wb") as f:

pickle.dump(rf, f)

##### 慢性肾病数据集定义模型

相关代码如下：

import pickle

with open("model\_kidney.pkl", "wb") as f:

pickle.dump(rf, f)

3.模型应用

##### 1)心脏病模型应用

#通过eli5得到各特征重要性

import eli5

from eli5.sklearn import PermutationImportance

perm = PermutationImportance(rf, random\_state=1).fit(x\_test.T, y\_test.T)

eli5.show\_weights(perm, feature\_names = x\_test.T.columns.tolist())

输出的各特征重要性如图2-11所示。

#正常人平均水平

average0\_count=np.multiply(average0,w)

average0\_sum=sum(average0\_count)

#病人平均水平

average1\_count=np.multiply(average1,w)

average1\_sum=sum(average1\_count)

#输出得到的数值

print(average1\_sum) #患者

print(average0\_sum) #正常人

将这个值保存，当用户使用时，判断出是患者还是正常人之后，根据比值大小定量判断具体情况。

##### 2)慢性肾病模型应用创新

通过eli5得到各特征重要性，输出的各特征重要性如图2-14所示。

import eli5

from eli5.sklearn import PermutationImportance

perm = PermutationImportance(rf, random\_state=1).fit(x\_test.T, y\_test.T)

eli5.show\_weights(perm, feature\_names = x\_test.T.columns.tolist())

import eli5 #for purmutation importance

from eli5.sklearn import PermutationImportance

perm = PermutationImportance(rf, random\_state=1).fit(X\_test, y\_test)

eli5.show\_weights(perm, feature\_names = X\_test.columns.tolist())

#求均值

#正常人平均水平

average0\_count=np.multiply(average0,w)

average0\_sum=sum(average0\_count)

#病人平均水平

average1\_count=np.multiply(average1,w)

average1\_sum=sum(average1\_count)

print(average1\_sum)#患者

print(average0\_sum)#正常人

#输出得到的数值

print(average1\_sum)#病人

print(average0\_sum)#正常人

将这个值保存，当用户使用时，判断是患者还是正常人之后，根据比值大小定量出具体情况。

2.3.2 药物推荐

1.数据预处理

UCI ML药品评论数据集来源：<https://www.kaggle.com/jessicali9530/kuc-hackathon-winter-2018> 。包括超20多万条不同用户在某一种症状下服用某药物后的评论，并根据效果从1~10进行打分。通过分析该数据集，可以对用户症状推荐大众认可的药物。

加载数据集和数据预处理，大部分通过Pandas实现，相关代码如下：

#导入相应库函数

import pandas as pd

#读取评论数据集

train = pd.read\_csv('../Thursday9 10 11/drugsComTrain\_raw.csv')

test = pd.read\_csv('../Thursday9 10 11/drugsComTest\_raw.csv')

会自动从csv数据源读取相应的数据

#通过Pandas的统计，全部评论数为

print('全部评论数：')

print(len(train))

print(len(test))

print('两端评分有：')

print(len(train))

print(len(test))

打分

train.rating.hist(bins=10)

plt.title('Distribution of Ratings')

plt.xlabel('Rating')

plt.ylabel('Count')

plt.xticks([i for i in range(1, 11)]);

#取出评分为1和10两端的数据

train=train[train.rating.isin([1,10])]

test=test[test.rating.isin([1,10])]

评论(review)中，句子两端有引号，编写函数将引号删除。

def remove\_enclosing\_quotes(s):

if s[0] == '"' and s[-1] == '"':

return s[1:-1]

else:

return s

#调用写好的函数，删除双引号

train.review = train.review.apply(remove\_enclosing\_quotes)

test.review = test.review.apply(remove\_enclosing\_quotes

发现一句话中经常出现不合时宜的符号，该数据集是网络爬虫爬取的，所以有很多字符表示成ASCII码，防止被误识别为分隔，使用正则表达式从审阅文本中删除这些符号。

import re

train.review = train.review.apply(lambda x: re.sub(r'&#\d+;',r'', x))

test.review = test.review.apply(lambda x: re.sub(r'&#\d+;',r'', x))

预测的标签是喜欢与不喜欢，但是drugName和condition种类很多，写进程序中可以简化工作量，所以需要将drugName和condition列前置到review中，并将完整的字符串保存为text列。

#定义函数

def combine\_text\_columns(data\_frame, text\_cols):

text\_data = data\_frame[text\_cols]

text\_data.fillna("", inplace=True)

return text\_data.apply(lambda x: " ".join(x), axis=1)

#将drugName和condition列前置到review中

text\_cols = ['drugName', 'condition', 'review']

train['text'] = combine\_text\_columns(train, text\_cols)

test['text'] = combine\_text\_columns(test, text\_cols)

CountVectorizer类将文本中的词语转换为词频矩阵。

#过滤规则，token的正则表达式

TOKENS\_ALPHANUMERIC = '[A-Za-z0-9]+(?=\\s+)'

#CountVectorizer对象的实例化，停用词选为english内置的英语停用词

vec\_alphanumeric = CountVectorizer(token\_pattern=TOKENS\_ALPHANUMERIC, ngram\_range=(1,2), lowercase=True, stop\_words='english', min\_df=2, max\_df=0.99)

#fit\_transform是fit和transform的组合，对部分数据先拟合fit，找到该part的整体指标，如均值、方差、最大值、最小值等，对trainData转换成transform，实现数据的标准化、归一化

X = vec\_alphanumeric.fit\_transform(train.text)

#1和10是两类，从5分开还是6分开无所谓，因为当前数据集中只有1分和10分

train['binary\_rating'] = train['rating'] > 5

y = train.binary\_rating

#使用Scikit-learn的train\_test\_split自动划分训练集和测试集

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state=42, stratify=y, test\_size=0.1)

#UCI ML药品评论预处理完成

2.模型训练及应用

相关代码如下：

#使用逻辑斯蒂回归训练模型

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state=42, stratify=y, test\_size=0.1)

clf\_lr = LogisticRegression(penalty='l2', C=100).fit(X\_train, y\_train)

#在测试集检验模型准确度

pred = clf\_lr.predict(X\_test)

#输出模型准确度

print("Accuracy on training set: {}".format(clf\_lr.score(X\_train, y\_train)))

print("Accuracy on test set: {}".format(clf\_lr.score(X\_test, y\_test)))

将评论经过数据预处理后，带入训练好的模型，得到评论感情分类。

pred\_0 = clf\_lr.predict(X\_0)

#输出最终判决结果

print(pred\_0)



图2-24 带入模型分析评论情感为支持或不支持

#输出结果到csv文件中

import csv

data =pred\_0

with open('medicine.csv','r') as csvFile: #此处的csv是源表

rows = csv.reader(csvFile)

with open('2.csv','w',newline='support') as f:#这里csv是最后输出得到的新表

writer = csv.writer(f)

i = 0

for row in rows:

row.append(data[i])

print(i)

i = i + 1

writer.writerow(row)

3.模型应用

help\_dict = {}

#unique方法不重复的记录所有症状，遍历

import csv

headers = ['condition','medicine\_1','medicine\_2','medicine\_3']

with open('cure.csv','a',newline='') as f:

f\_csv = csv.writer(f)

f\_csv.writerow(headers)

for i in train.condition.unique():

temp\_ls = []

#遍历这个症状所提到，且被认同的药物

for j in train[train.condition == i & train.support==True ].drugName.unique():

#如果这种药物至少10个人提及，则记录下来

if np.sum(train.drugName == j) >= 10:

temp\_ls.append((j, np.sum(train[train.drugName == j].rating) / np.sum(train.drugName == j)))

#针对症状i,从好到坏将刚刚提到的药进行排名

help\_dict[i] = pd.DataFrame(data=temp\_ls, columns=['drug', 'average\_rating']).sort\_values(by='average\_rating', ascending=False).reset\_index(drop=True) rows=[(i,help\_dict[i].iloc[0:1].drug,help\_dict[i].iloc[1:2].drug,help\_dict[i].iloc[2:3].drug)]

f\_csv.writerows(rows)

f.close()

#最终完成遍历时，在编译界面有一个反馈

print('ok')

2.3.3 模块应用

1.模型导入

client\_result=rf.predict(client\_x)

print('这就是分类预测结果')

print(client\_result)

2.相关代码

##### 1)模型预测

相关代码如下：

#心脏病预测模型建模

#导入所用库函数及数据集

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.linear\_model import LogisticRegression

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

import os

#读入数据集中数据

df = pd.read\_csv("C:/Users/Administrator/Desktop/dasanxia/Thursday9 10 11/heart.csv")

#输出读入的数据

df.head()

#检查是否有缺省值

df.loc[(df['age'].isnull()) |

(df['sex'].isnull()) |

(df['cp'].isnull()) |

(df['trestbps'].isnull()) |

(df['chol'].isnull()) |

(df['fbs'].isnull()) |

(df['restecg'].isnull()) |

(df['thalach'].isnull()) |

(df['exang'].isnull()) |

(df['oldpeak'].isnull()) |

(df['slope'].isnull()) |

(df['ca'].isnull()) |

(df['target'].isnull())]

#通过绘图方式观察数据，能够更好的观察是否有错误

sns.pairplot(df.dropna(), hue='target')

#对血液中胆固醇含量绘制数据分布图

df['chol'].hist()

#对静息血压绘制数据分布图

df['trestbps'].hist()

#将类别变量转换为伪变量

a = pd.get\_dummies(df['cp'], prefix = "cp")

b = pd.get\_dummies(df['thal'], prefix = "thal")

c = pd.get\_dummies(df['slope'], prefix = "slope")

frames = [df, a, b, c]

df = pd.concat(frames, axis = 1)

#数据集可视化

df.head()

#将原来的类别变量删掉，只保留伪变量

df = df.drop(columns = ['cp', 'thal', 'slope'])

#数据集可视化

df.head()

#数据预处理完成，选择、训练并保存模型

#target是标签

y = df.target.values

x\_data = df.drop(['target'], axis = 1) #丢下最后一行target

#划分训练集与测试集

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x\_data,y,test\_size = 0.2,random\_state=0)

x\_train = x\_train.T

y\_train = y\_train.T

x\_test = x\_test.T

y\_test = y\_test.T

#创建数组，带入随机森林模型中迭代

import numpy as np

num=np.zeros(20,int)

for i in range(0,20):

num[i]=i+990

print(num)

#寻找最优参数

from sklearn.ensemble import RandomForestClassifier

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

tuned\_parameters = [{'n\_estimators':num,

'class\_weight':[None,{0: 0.33,1:0.67},'balanced'],'random\_state':[1]}]

rf = GridSearchCV(RandomForestClassifier(), tuned\_parameters, cv=10,scoring='f1')

rf.fit(x\_train.T, y\_train.T)

#输出找到的最优参数

print('Best parameters:')

print(rf.best\_params\_)

rf\_best = rf.best\_estimator\_

#带入最优参数的随机森林模型

accuracies = {}

rf\_best.fit(x\_train.T, y\_train.T)

acc = rf.score(x\_test.T,y\_test.T)\*100

accuracies['Random Forest'] = acc

#输出模型准确率

print("Random Forest Algorithm Accuracy Score : {:.2f}%".format(acc))

#绘制混淆矩阵

y\_head\_rf = rf\_best.predict(x\_test.T)

from sklearn.metrics import confusion\_matrix

cm\_rf = confusion\_matrix(y\_test,y\_head\_rf)

#图像大小4\*4

plt.figure(figsize=(4,4))

plt.title("Random Forest Confusion Matrix")

sns.heatmap(cm\_rf,annot=True,cmap="Blues",fmt="d",cbar=False, annot\_kws={"size": 24})

plt.show()

#绘制ROC曲线

from sklearn.metrics import roc\_curve, auc

fpr, tpr, thresholds = roc\_curve(y\_test, y\_head\_rf)

fig, ax = plt.subplots()

ax.plot(fpr, tpr)

ax.plot([0, 1], [0, 1], transform=ax.transAxes, ls="--", c=".3")

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.0])

plt.rcParams['font.size'] = 12

plt.title('ROC curve for diabetes classifier')

plt.xlabel('False Positive Rate (1 - Specificity)')

plt.ylabel('True Positive Rate (Sensitivity)')

plt.grid(True)

#ROC曲线图的面积

auc(fpr, tpr)

#慢性肾病数据集训练模型建模

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

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

from sklearn.metrics import roc\_curve, auc, confusion\_matrix, classification\_report,accuracy\_score

from sklearn.ensemble import RandomForestClassifier

import warnings

warnings.filterwarnings('ignore')

#%matplotlib inline

#读入数据集并可视化

df = pd.read\_csv('C:/Users/Administrator/Desktop/dasanxia/Thursday9 10 11/kidney\_disease.csv')

df.head()

#数据预处理，将类别变量转换为伪变量

df[['htn','dm','cad','pe','ane']] = df[['htn','dm','cad','pe','ane']].replace(to\_replace={'yes':1,'no':0})

df[['rbc','pc']]=df[['rbc','pc']].replace(to\_replace={'abnormal':1,'normal':0})

df[['pcc','ba']]=df[['pcc','ba']].replace(to\_replace={'present':1,'notpresent':0})

df[['appet']]=df[['appet']].replace(to\_replace={'good':1,'poor':0,'no':np.nan})

df['classification']=df['classification'].replace(to\_replace={'ckd':1.0,'ckd\t':1.0,'notckd':0.0,'no':0.0})

df.rename(columns={'classification':'class'},inplace=True)

#进一步清洗

df['pe'] = df['pe'].replace(to\_replace='good',value=0)

df['appet']=df['appet'].replace(to\_replace='no',value=0)

df['cad']=df['cad'].replace(to\_replace='\tno',value=0)

df['dm']=df['dm'].replace(to\_replace={'\tno':0,'\tyes':1,' yes':1, '':np.nan})

df.drop('id',axis=1,inplace=True)

df.head()

#列出所有null的数据

df.loc[(df['age'].isnull()) |

(df['bp'].isnull()) |

(df['sg'].isnull()) |

(df['al'].isnull()) |

(df['su'].isnull()) |

(df['rbc'].isnull()) |

(df['pc'].isnull()) |

(df['pcc'].isnull()) |

(df['ba'].isnull()) |

(df['bgr'].isnull()) |

(df['bu'].isnull()) |

(df['sc'].isnull()) |

(df['sod'].isnull()) |

(df['pot'].isnull()) |

(df['hemo'].isnull()) |

(df['htn'].isnull()) |

(df['dm'].isnull()) |

(df['cad'].isnull()) |

(df['appet'].isnull()) |

(df['pe'].isnull()) |

(df['ane'].isnull()) |

(df['class'].isnull())]

#出现空缺值，采用均值归一法，填补缺失值

#病人均值

average0\_age = df.loc[df['class'] ==True, 'age'].mean()

average0\_bp = df.loc[df['class'] == True, 'bp'].mean()

average0\_sg = df.loc[df['class'] == True, 'sg'].mean()

average0\_al = df.loc[df['class'] == True, 'al'].mean()

average0\_su = df.loc[df['class'] == True, 'su'].mean()

average0\_rbc = df.loc[df['class'] == True, 'rbc'].mean()

average0\_pc = df.loc[df['class'] == True, 'pc'].mean()

average0\_pcc = df.loc[df['class'] == True, 'pcc'].mean()

average0\_ba = df.loc[df['class'] == True, 'ba'].mean()

average0\_bgr = df.loc[df['class'] == True, 'bgr'].mean()

average0\_bu = df.loc[df['class'] == True, 'bu'].mean()

average0\_sc = df.loc[df['class'] == True, 'sc'].mean()

average0\_sod = df.loc[df['class'] == True, 'sod'].mean()

average0\_pot = df.loc[df['class'] == True, 'pot'].mean()

average0\_hemo = df.loc[df['class'] == True, 'hemo'].mean()

average0\_htn = df.loc[df['class'] == True, 'htn'].mean()

average0\_dm = df.loc[df['class'] == True, 'dm'].mean()

average0\_cad = df.loc[df['class'] == True, 'cad'].mean()

average0\_appet = df.loc[df['class'] ==True, 'appet'].mean()

average0\_pe = df.loc[df['class'] == True, 'pe'].mean()

average0\_ane = df.loc[df['class'] == True, 'ane'].mean()

#正常人均值

average1\_age = df.loc[df['class'] == False, 'age'].mean()

average1\_bp = df.loc[df['class'] == False, 'bp'].mean()

average1\_sg = df.loc[df['class'] == False, 'sg'].mean()

average1\_al = df.loc[df['class'] == False, 'al'].mean()

average1\_su = df.loc[df['class'] == False, 'su'].mean()

average1\_rbc = df.loc[df['class'] == False, 'rbc'].mean()

average1\_pc = df.loc[df['class'] == False, 'pc'].mean()

average1\_pcc = df.loc[df['class'] == False, 'pcc'].mean()

average1\_ba = df.loc[df['class'] == False, 'ba'].mean()

average1\_bgr = df.loc[df['class'] == False, 'bgr'].mean()

average1\_bu = df.loc[df['class'] == False, 'bu'].mean()

average1\_sc = df.loc[df['class'] == False, 'sc'].mean()

average1\_sod = df.loc[df['class'] == False, 'sod'].mean()

average1\_pot = df.loc[df['class'] == False, 'pot'].mean()

average1\_hemo = df.loc[df['class'] == False, 'hemo'].mean()

average1\_htn = df.loc[df['class'] == False, 'htn'].mean()

average1\_dm = df.loc[df['class'] == False, 'dm'].mean()

average1\_cad = df.loc[df['class'] == False, 'cad'].mean()

average1\_appet = df.loc[df['class'] == False, 'appet'].mean()

average1\_pe = df.loc[df['class'] == False, 'pe'].mean()

average1\_ane = df.loc[df['class'] == False, 'ane'].mean()

#如果为null，则取均值

df.loc[(df['class'] ==True) &(df['age'].isnull()),'age'] = average0\_age

df.loc[(df['class'] ==True) &(df['bp'].isnull()),'bp'] = average0\_bp

df.loc[(df['class'] ==True) &(df['sg'].isnull()),'sg'] = average0\_sg

df.loc[(df['class'] ==True) &(df['al'].isnull()),'al'] = average0\_al

df.loc[(df['class'] ==True) &(df['su'].isnull()),'su'] = average0\_su

df.loc[(df['class'] ==True) &(df['rbc'].isnull()),'rbc'] = average0\_rbc

df.loc[(df['class'] ==True) &(df['pc'].isnull()),'pc'] = average0\_pc

df.loc[(df['class'] ==True) &(df['pcc'].isnull()),'pcc'] = average0\_pcc

df.loc[(df['class'] ==True) &(df['ba'].isnull()),'ba'] = average0\_ba

df.loc[(df['class'] ==True) &(df['bgr'].isnull()),'bgr'] = average0\_bgr

df.loc[(df['class'] ==True) &(df['bu'].isnull()),'bu'] = average0\_bu

df.loc[(df['class'] ==True) &(df['sc'].isnull()),'sc'] = average0\_sc

df.loc[(df['class'] ==True) &(df['sod'].isnull()),'sod'] = average0\_sod

df.loc[(df['class'] ==True) &(df['pot'].isnull()),'pot'] = average0\_pot

df.loc[(df['class'] ==True) &(df['hemo'].isnull()),'hemo']=average0\_hemo

df.loc[(df['class'] ==True) &(df['htn'].isnull()),'htn'] = average0\_htn

df.loc[(df['class'] ==True) &(df['dm'].isnull()),'dm'] = average0\_dm

df.loc[(df['class'] ==True) &(df['cad'].isnull()),'cad'] = average0\_cad

df.loc[(df['class'] ==True) &(df['appet'].isnull()),'appet'] = average0\_appet

df.loc[(df['class'] ==True)&(df['pe'].isnull()),'pe'] = average0\_pe

df.loc[(df['class'] ==True) &(df['ane'].isnull()),'ane'] = average0\_ane

df.loc[(df['class'] ==False) &(df['age'].isnull()),'age'] = average1\_age

df.loc[(df['class'] ==False) &(df['bp'].isnull()),'bp'] = average1\_bp

df.loc[(df['class'] ==False) &(df['sg'].isnull()),'sg'] = average1\_sg

df.loc[(df['class'] ==False) &(df['al'].isnull()),'al'] = average1\_al

df.loc[(df['class'] ==False) &(df['su'].isnull()),'su'] = average1\_su

df.loc[(df['class'] ==False) &(df['rbc'].isnull()),'rbc'] = average1\_rbc

df.loc[(df['class'] ==False) &(df['pc'].isnull()),'pc'] = average1\_pc

df.loc[(df['class'] ==False) &(df['pcc'].isnull()),'pcc'] = average1\_pcc

df.loc[(df['class'] ==False) &(df['ba'].isnull()),'ba'] = average1\_ba

df.loc[(df['class'] ==False) &(df['bgr'].isnull()),'bgr'] = average1\_bgr

df.loc[(df['class'] ==False) &(df['bu'].isnull()),'bu'] = average1\_bu

df.loc[(df['class'] ==False) &(df['sc'].isnull()),'sc'] = average1\_sc

df.loc[(df['class'] ==False) &(df['sod'].isnull()),'sod'] = average1\_sod

df.loc[(df['class'] ==False) &(df['pot'].isnull()),'pot'] = average1\_pot

df.loc[(df['class'] ==False) &(df['hemo'].isnull()),'hemo'] = average1\_hemo

df.loc[(df['class'] ==False) &(df['htn'].isnull()),'htn'] = average1\_htn

df.loc[(df['class'] ==False) &(df['dm'].isnull()),'dm'] = average1\_dm

df.loc[(df['class'] ==False) &(df['cad'].isnull()),'cad'] = average1\_cad

df.loc[(df['class'] ==False) &(df['appet'].isnull()),'appet'] = average1\_appet

df.loc[(df['class'] ==False) &(df['pe'].isnull()),'pe'] = average1\_pe

df.loc[(df['class'] ==False) &(df['ane'].isnull()),'ane'] = average1\_ane

#重新检查是否有缺省值

df.loc[(df['age'].isnull()) |

(df['bp'].isnull()) |

(df['sg'].isnull()) |

(df['al'].isnull()) |

(df['su'].isnull()) |

(df['rbc'].isnull()) |

(df['pc'].isnull()) |

(df['pcc'].isnull()) |

(df['ba'].isnull()) |

(df['bgr'].isnull()) |

(df['bu'].isnull()) |

(df['sc'].isnull()) |

(df['sod'].isnull()) |

(df['pot'].isnull()) |

(df['hemo'].isnull()) |

(df['htn'].isnull()) |

(df['dm'].isnull()) |

(df['cad'].isnull()) |

(df['appet'].isnull()) |

(df['pe'].isnull()) |

(df['ane'].isnull()) |

(df['class'].isnull())]

#划分训练集测试集

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df.iloc[:,:-1], df['class'], test\_size = 0.33, random\_state=44,stratify= df['class'] )

print(X\_train.shape)

print(X\_test.shape)

#寻找随机森林最优参数

tuned\_parameters = [{'n\_estimators':[7,8,9,10,11,12,13,14,15,16],'max\_depth':[2,3,4,5,6,None],'class\_weight':[None,{0: 0.33,1:0.67},'balanced'],'random\_state':[42]}]

clf = GridSearchCV(RandomForestClassifier(), tuned\_parameters, cv=10,scoring='f1')

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

#输出最佳参数

print('Best parameters:')

print(clf.best\_params\_)

clf\_best = clf.best\_estimator\_

#将最优参数代入随机森林模型

accuracies = {}

rf = RandomForestClassifier(class\_weight=None, max\_depth= 6,n\_estimators = 7, random\_state = 42)

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

#计算模型准确率

acc = rf.score(X\_test, y\_test)\*100

accuracies['Random Forest'] = acc

print("Random Forest Algorithm Accuracy Score : {:.2f}%".format(acc))

#药物评论情感分析建模

#导入库函数

import warnings

warnings.filterwarnings('ignore')

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from matplotlib import style; style.use('ggplot')

import re

import xgboost as xgb

from nltk.sentiment.vader import SentimentIntensityAnalyzer

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

from sklearn.pipeline import Pipeline

from sklearn.ensemble import RandomForestClassifier

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import confusion\_matrix, classification\_report, roc\_curve, roc\_auc\_score

from sklearn.naive\_bayes import MultinomialNB, GaussianNB

from sklearn.feature\_extraction.text import HashingVectorizer

from sklearn.feature\_selection import chi2, SelectKBest

from sklearn.preprocessing import StandardScaler

from sklearn.svm import SVC

from wordcloud import WordCloud, STOPWORDS

from keras.models import Sequential

from keras.layers import Dense, LSTM, Embedding

from keras.utils import to\_categorical

#读入训练集和测试集

train = pd.read\_csv('C:/Users/Administrator/Desktop/dasanxia/Thursday9 10 11/drugsComTrain\_raw.csv')

test = pd.read\_csv('C:/Users/Administrator/Desktop/dasanxia/Thursday9 10 11/drugsComTest\_raw.csv')

#输出可视化

print('全部评论数：')

print(len(train))

print(len(test))

#仅保留评分为1和10分两端的评论

X\_train=train[train.rating.isin([1,10])]

X\_train.head()

X\_test=test[test.rating.isin([1,10])]

X\_test.head()

#编写去除评论中引号的函数

def remove\_enclosing\_quotes(s):

if s[0] == '"' and s[-1] == '"':

return s[1:-1]

else:

return s

#调用函数

train.review = train.review.apply(remove\_enclosing\_quotes)

test.review = test.review.apply(remove\_enclosing\_quotes)

#用正则表达式去除乱码，防止对后续分隔句子造成影响

import re

train.review = train.review.apply(lambda x: re.sub(r'&#\d+;',r'', x))

test.review = test.review.apply(lambda x: re.sub(r'&#\d+;',r'', x))

#编写函数，将症状，药物写进评论中，拼成一个整体

def combine\_text\_columns(data\_frame, text\_cols):

text\_data = data\_frame[text\_cols]

text\_data.fillna("", inplace=True)

return text\_data.apply(lambda x: " ".join(x), axis=1)

#调用函数

text\_cols = ['drugName', 'condition', 'review']

train['text'] = combine\_text\_columns(train, text\_cols)

test['text'] = combine\_text\_columns(test, text\_cols)

#过滤规则，token的正则表达式

TOKENS\_ALPHANUMERIC = '[A-Za-z0-9]+(?=\\s+)'

#CountVectorizer 类将文本中的词语转换为词频矩阵

vec\_alphanumeric = CountVectorizer(token\_pattern=TOKENS\_ALPHANUMERIC, ngram\_range=(1,2), lowercase=True, stop\_words='english', min\_df=2, max\_df=0.99)

#转换transform，从而实现数据的标准化、归一化

X = vec\_alphanumeric.fit\_transform(train.text)

#将1和10分评论二分类，归位两堆

train['binary\_rating'] = train['rating'] > 5

y = train.binary\_rating

#划分训练集和测试集

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state=42, stratify=y, test\_size=0.1)

#逻辑回归训练模型

clf\_lr = LogisticRegression(penalty='l2', C=100).fit(X\_train, y\_train)

pred = clf\_lr.predict(X\_test)

#模型在训练集准确率

print("Accuracy on training set: {}".format(clf\_lr.score(X\_train, y\_train)))

#模型在测试集准确率

print("Accuracy on test set: {}".format(clf\_lr.score(X\_test, y\_test)))

#绘制混淆矩阵

print("Confusion Matrix")

print(confusion\_matrix(y\_test, pred))

print(classification\_report(y\_test, pred))

##### 2)模型应用创新

相关代码如下：

#得到心脏病数据集中特征重要性

import eli5

from eli5.sklearn import PermutationImportance

perm = PermutationImportance(rf, random\_state=1).fit(x\_test.T, y\_test.T)

eli5.show\_weights(perm, feature\_names = x\_test.T.columns.tolist())

#得到正常人和患者均值

df.groupby('target').mean()

#正常人平均水平

average0\_count=np.multiply(average0,w)

average0\_sum=sum(average0\_count)

#病人平均水平

average1\_count=np.multiply(average1,w)

average1\_sum=sum(average1\_count)

#得到慢性肾病数据集中特征重要性

import eli5 #for purmutation importance

from eli5.sklearn import PermutationImportance

perm = PermutationImportance(rf, random\_state=1).fit(X\_test, y\_test)

eli5.show\_weights(perm, feature\_names = X\_test.columns.tolist())

df.groupby('class').mean()

#病人平均水平

average0\_count=np.multiply(average0,w)

average0\_sum=sum(average0\_count)

#正常人平均水平

average1\_count=np.multiply(average1,w)

average1\_sum=sum(average1\_count)

#对感情不明确的进行情感分析

train\_0 = pd.read\_csv('C:/Users/Administrator/Desktop/dasanxia/Thursday9 10 11/drugsComTrain\_raw.csv')

test\_0 = pd.read\_csv('C:/Users/Administrator/Desktop/dasanxia/Thursday9 10 11/drugsComTest\_raw.csv')

#仅读入打分为2~9的评论

train\_0=train\_0[train\_0.rating.isin([2,3,4,5,6,7,8,9])]

test\_0=test\_0[test\_0.rating.isin([2,3,4,5,6,7,8,9])]

#去除双引号

train\_0.review = train\_0.review.apply(remove\_enclosing\_quotes)

test\_0.review = test\_0.review.apply(remove\_enclosing\_quotes)

#去除特殊字符

train\_0.review = train\_0.review.apply(lambda x: re.sub(r'&#\d+;',r'', x))

test\_0.review = test\_0.review.apply(lambda x: re.sub(r'&#\d+;',r'', x))

#将症状、药物与评论三者融为一段文字

train\_0['text'] = combine\_text\_columns(train\_0, text\_cols)

test\_0['text'] = combine\_text\_columns(test\_0, text\_cols)

#数据归一化

X\_0 = vec\_alphanumeric.fit\_transform(train.text)

#调用模型进行预测

pred\_0 = clf\_lr.predict(X\_0)

#输出预测结果

print(pred\_0)

##### 3)用户接口及界面可视化代码

本部分为主界面GUI设计及子界面调用。

#导入库函数

from PyQt5 import QtCore, QtGui, QtWidgets

from PyQt5.QtCore import QCoreApplication

#GUI界面大小，按键设置

class Ui\_Dialog(object):

def setupUi(self, Dialog):

Dialog.setObjectName("Dialog")

Dialog.resize(1600, 1000)

Dialog.setFixedSize(1600, 1000)

self.label = QtWidgets.QLabel(Dialog)

self.label.setGeometry(QtCore.QRect(680, 60, 250, 61))

self.label.setObjectName("label")

self.label.setStyleSheet('font-size:40px')

self.pushButton = QtWidgets.QPushButton(Dialog)

self.pushButton.setGeometry(QtCore.QRect(720, 300, 150, 50))

self.pushButton.setObjectName("pushButton")

self.pushButton.setStyleSheet('font-size:30px')

self.pushButton\_2 = QtWidgets.QPushButton(Dialog)

self.pushButton\_2.setGeometry(QtCore.QRect(720, 500, 150, 50))

self.pushButton\_2.setObjectName("pushButton\_2")

self.pushButton\_2.setStyleSheet('font-size:30px')

self.pushButton\_3 = QtWidgets.QPushButton(Dialog)

self.pushButton\_3.setGeometry(QtCore.QRect(720, 800, 150, 50))

self.pushButton\_3.setObjectName("pushButton\_3")

self.pushButton\_3.setStyleSheet('font-size:30px')

self.pushButton\_3.clicked.connect(QCoreApplication.instance().quit)

self.retranslateUi(Dialog)

QtCore.QMetaObject.connectSlotsByName(Dialog)

#为按键起名字

def retranslateUi(self, Dialog):

\_translate = QtCore.QCoreApplication.translate

Dialog.setWindowTitle(\_translate("Dialog", "智能健康助手"))

self.label.setText(\_translate("Dialog", "智能健康助手"))

self.pushButton.setText(\_translate("Dialog", "健康预测"))

self.pushButton\_2.setText(\_translate("Dialog", "药物推荐"))

self.pushButton\_3.setText(\_translate("Dialog", "退出程序"))

#调用机器学习模型，对用户的数据进行分析

#导入库函数

import csv

import pandas as pd

import pickle

import numpy as np

from PyQt5 import QtCore, QtGui, QtWidgets

from databank import result

#GUI界面大小，按键设置

class Ui\_Dialog(object):

def setupUi(self, Dialog):

Dialog.setObjectName("Dialog")

Dialog.resize(1600, 1000)

Dialog.setFixedSize(1600,1000)

self.label = QtWidgets.QLabel(Dialog)

self.label.setGeometry(QtCore.QRect(650,40,300,80)) #标签位置及大小

self.label.setTextFormat(QtCore.Qt.AutoText)

self.label.setAlignment(QtCore.Qt.AlignCenter)

self.label.setObjectName("label")

self.label.setStyleSheet('font-size:40px')

self.age = QtWidgets.QLabel(Dialog)

self.age.setGeometry(QtCore.QRect(700, 200, 200, 40))

self.age.setObjectName("age")

self.age.setStyleSheet('font-size:30px')

self.ageinput = QtWidgets.QLineEdit(Dialog)

self.ageinput.setGeometry(QtCore.QRect(850, 200, 50, 40))

self.ageinput.setObjectName("ageinput")

agelimit = QtCore.QRegExp("[1-9][0-9]{1,2}")

age\_validator = QtGui.QRegExpValidator(agelimit, self.ageinput)

self.ageinput.setValidator(age\_validator)

font = QtGui.QFont()

font.setPointSize(20)

self.ageinput.setFont(font)

self.sex = QtWidgets.QLabel(Dialog)

self.sex.setGeometry(QtCore.QRect(700, 300, 200, 40))

self.sex.setObjectName("sex")

self.sex.setStyleSheet('font-size:30px')

self.sexinput = QtWidgets.QComboBox(Dialog)

self.sexinput.setGeometry(QtCore.QRect(850, 300, 60, 45))

self.sexinput.setObjectName("sexinput")

self.sexinput.addItem("")

self.sexinput.addItem("")

self.sexinput.setStyleSheet('font-size:30px')

self.taste = QtWidgets.QLabel(Dialog)

self.taste.setGeometry(QtCore.QRect(700, 400, 200, 40))

self.taste.setObjectName("taste")

self.taste.setStyleSheet('font-size:30px')

self.tasteinput = QtWidgets.QComboBox(Dialog)

self.tasteinput.setGeometry(QtCore.QRect(850, 400, 100, 45))

self.tasteinput.setObjectName("tasteinput")

self.tasteinput.addItem("")

self.tasteinput.addItem("")

self.tasteinput.setStyleSheet('font-size:30px')

self.pushButton = QtWidgets.QPushButton(Dialog)

self.pushButton.setGeometry(QtCore.QRect(750, 800, 100, 40))

self.pushButton.setObjectName("pushButton")

self.pushButton.setStyleSheet('font-size:30px')

self.pushButton.clicked.connect(self.do)

self.Button = QtWidgets.QPushButton(Dialog)

self.Button.setGeometry(QtCore.QRect(700, 700, 200, 40))

self.Button.setObjectName("Button")

self.Button.setStyleSheet('font-size:30px')

self.Button.clicked.connect(self.get)

self.Button2 = QtWidgets.QPushButton(Dialog)

self.Button2.setGeometry(QtCore.QRect(750, 950, 100, 40))

self.Button2.setObjectName("Button")

self.Button2.setStyleSheet('font-size:30px')

self.label\_2 = QtWidgets.QLabel(Dialog)

self.label\_2.setGeometry(QtCore.QRect(900, 700, 200, 40))

self.label\_2.setStyleSheet('font-size:30px')

self.label\_2.setObjectName("label\_2")

self.progressBar = QtWidgets.QProgressBar(Dialog)

self.progressBar.setGeometry(QtCore.QRect(760, 900, 118, 23))

self.progressBar.setProperty("value", 0)

self.progressBar.setObjectName("progressBar")

self.retranslateUi(Dialog)

QtCore.QMetaObject.connectSlotsByName(Dialog)

#读入用户输入数据

def get(self):

age = self.ageinput.text()

sex = self.sexinput.currentIndex()

taste = self.tasteinput.currentIndex()

a = [age,sex,145,233,1,0,150,0,2.3,0,0,0,0,1,0,1,0,0,1,0,0]

b=[age,80,1.02,1,0,0.439252336,0,0,0,121,36,1.2,133.9017857,4.878443114,15.4,1,1,0,taste,0,0]

#将用户数据输出到csv文件，并进行定性化分析

with open('./databank/test.csv', "a",newline='') as file:

csv\_file = csv.writer(file)

csv\_file.writerow(a)

file.close()

with open('./databank/test2.csv', "a",newline='') as file:

csv\_file = csv.writer(file)

csv\_file.writerow(b)

file.close()

self.label\_2.setText(QtCore.QCoreApplication.translate("Dialog", "写入成功"))

self.progressBar.setProperty("value", 0)

#对用户数据定量化分析

def caculate1(self,client1,client\_result1,):

#判断病情，数据来源于模型

average1\_sum = 2.94673976 #病人

average0\_sum = 3.12316913 #正常人

#权重矩阵

w = np.ones((21, 1))

w[0, 0] = 0.0066 #client\_x['age']

w[1, 0] = 0.059 #client\_x['ca']

w[2, 0] = 0.0033 #client\_x['chol']

w[3, 0] = 0.0590 #client\_x['cp\_0']

w[4, 0] = 0 #client\_x['cp\_1']

w[5, 0] = 0.0197 #client\_x['cp\_2']

w[6, 0] = 0.0131 #client\_x['cp\_3']

w[7, 0] = -0.0033 #client\_x['exang']

w[8, 0] = -0.0033 #client\_x['fbs']

w[9, 0] = 0.0098 #client\_x['oldpeak']

w[10, 0] = 0.0131 #client\_x['restecg']

w[11, 0] = 0.0131 #client\_x['trestbps']

w[12, 0] = 0.0098 #client\_x['sex']

w[13, 0] = 0 #client\_x['slope\_0']

w[14, 0] = 0.0033 #client\_x['slope\_1']

w[15, 0] = 0.0066 #client\_x['slope\_2']

w[16, 0] = 0 #client\_x['thal\_0']

w[17, 0] = 0 #client\_x['thal\_1']

w[18, 0] = 0.0361 #client\_x['thal\_2']

w[19, 0] = 0.0131 #client\_x['thal\_3']

w[20, 0] = 0 #client\_x['thalach']

#用户数据转化为矩阵

client\_message = np.ones((21, 1))

client\_message[0, 0] = client1['age']

client\_message[1, 0] = client1['ca']

client\_message[2, 0] = client1['chol']

client\_message[3, 0] = client1['cp\_0']

client\_message[4, 0] = client1['cp\_1']

client\_message[5, 0] = client1['cp\_2']

client\_message[6, 0] = client1['cp\_3']

client\_message[7, 0] = client1['exang']

client\_message[8, 0] = client1['fbs']

client\_message[9, 0] = client1['oldpeak']

client\_message[10, 0] = client1['restecg']

client\_message[11, 0] = client1['trestbps']

client\_message[12, 0] = client1['sex']

client\_message[13, 0] = client1['slope\_0']

client\_message[14, 0] = client1['slope\_1']

client\_message[15, 0] = client1['slope\_2']

client\_message[16, 0] = client1['thal\_0']

client\_message[17, 0] = client1['thal\_1']

client\_message[18, 0] = client1['thal\_2']

client\_message[19, 0] = client1['thal\_3']

client\_message[20, 0] = client1['thalach']

client\_count = np.multiply(client\_message, w)

client\_sum = sum(client\_count)

#判断用户是否为病人

if client\_result1 == 1:

client\_index = client\_sum / average1\_sum

return(client\_index)

else:

client\_index = client\_sum / average0\_sum

return(client\_index)

#在判断用户是否为病人后继续定量化分析

def caculate2(self,client2,client\_result2,):

#判断病情，数据来源于模型

average1\_sum = 10.06672071 #病人

average0\_sum = 12.62982294 #正常人

#权重矩阵

w = np.ones((21, 1))

w[0, 0] = 0.002 #client\_x['age']

w[1, 0] = 0.051 #client\_x['bp']

w[2, 0] = 0.0545 #client\_x['sg']

w[3, 0] = 0.024 #client\_x['al']

w[4, 0] = 0 #client\_x['su']

w[5, 0] = 0.179 #client\_x['rbc']

w[6, 0] = 0 #client\_x['pc']

w[7, 0] = 0 #client\_x['pcc']

w[8, 0] = 0 #client\_x['ba']

w[9, 0] = 0.028 #client\_x['bgr']

w[10, 0] = 0.019 #client\_x['bu']

w[11, 0] = 0 #client\_x['sc']

w[12, 0] = 0.0025 #client\_x['sod']

w[13, 0] = 0 #client\_x['pot']

w[14, 0] = 0.1505 #client\_x['hemo']

w[15, 0] = 0.08 #client\_x['htn']

w[16, 0] = 0.025 #client\_x['dm']

w[17, 0] = 0 #client\_x['cad']

w[18, 0] = -0.0005 #client\_x['appet']

w[19, 0] = 0 #client\_x['pe']

w[20, 0] = -0.0005 #client\_x['ane']

client\_message = np.ones((21, 1))

client\_message[0, 0] = client2['age']

client\_message[1, 0] = client2['bp']

client\_message[2, 0] = client2['sg']

client\_message[3, 0] = client2['al']

client\_message[4, 0] = client2['su']

client\_message[5, 0] = client2['rbc']

client\_message[6, 0] = client2['pc']

client\_message[7, 0] = client2['pcc']

client\_message[8, 0] = client2['ba']

client\_message[9, 0] = client2['bgr']

client\_message[10, 0] = client2['bu']

client\_message[11, 0] = client2['sc']

client\_message[12, 0] = client2['sod']

client\_message[13, 0] = client2['pot']

client\_message[14, 0] = client2['hemo']

client\_message[15, 0] = client2['htn']

client\_message[16, 0] = client2['dm']

client\_message[17, 0] = client2['cad']

client\_message[18, 0] = client2['appet']

client\_message[19, 0] = client2['pe']

client\_message[20, 0] = client2['ane']

client\_count = np.multiply(client\_message, w)

client\_sum = sum(client\_count)

#print(client\_sum)调试代码

if client\_result2 == 1:

client\_index = client\_sum / average1\_sum

return(client\_index)

else:

client\_index = client\_sum / average0\_sum

return(client\_index)

#输出模型结果，可视化

def do(self):

self.label\_2.setText(QtCore.QCoreApplication.translate("Dialog", ""))

#QtWidgets.QMessageBox.about(None, "Warning", "跳转成功")调试代码

client = pd.read\_csv("./databank/test.csv")

#QtWidgets.QMessageBox.about(None, "Warning", "打开csv成功")调试代码

with open("./databank/model.pkl", "rb") as f:

rf = pickle.load(f)

list = []

#QtWidgets.QMessageBox.about(None, "Warning", "打开模型成功")调试代码

print(client.shape[0] - 1)

for i in range(0, client.shape[0] - 1):

list.append(i)

client = client.drop(list)

print(client)

#QtWidgets.QMessageBox.about(None, "Warning", "数据载入成功")调试代码

client\_result = rf.predict(client)

#QtWidgets.QMessageBox.about(None, "Warning", "预测成功")调试代码

print('这就是分类预测结果')

print(client\_result)

self.progressBar.setProperty("value", 25)

self.score1 = self.caculate1(client,client\_result)

print("得分结果：", self.score1)

self.progressBar.setProperty("value", 50)

client2 = pd.read\_csv("./databank/test2.csv")

with open("./databank/model\_kidney.pkl", "rb") as f:

rf2 = pickle.load(f)

list2 = []

for i in range(0, client2.shape[0] - 1):

list2.append(i)

client2 = client2.drop(list)

client2\_result = rf.predict(client2)

print('这就是分类预测结果')

print(client2\_result)

self.progressBar.setProperty("value", 75)

self.score2 = self.caculate2(client2, client2\_result)

print("得分结果：",self.score2)

self.progressBar.setProperty("value", 100)

self.c\_widget = QtWidgets.QWidget()

self.c = result.Ui\_Dialog() self.c.setupUi(client\_result[0],client2\_result[0],self.score1[0],self.score2[0],self.c\_widget)

self.c.pushButton.clicked.connect(self.c\_widget.close)

self.c\_widget.show()

#子界面文字设计

def retranslateUi(self, Dialog):

\_translate = QtCore.QCoreApplication.translate

Dialog.setWindowTitle(\_translate("Dialog", "健康预测系统"))

self.label.setText(\_translate("Dialog", "健康预测系统"))

self.age.setText(\_translate("Dialog", "您的年龄："))

self.sex.setText(\_translate("Dialog", "您的性别："))

self.sexinput.setItemText(0, \_translate("Dialog", "男"))

self.sexinput.setItemText(1, \_translate("Dialog", "女"))

self.taste.setText(\_translate("Dialog", "最近食欲："))

self.tasteinput.setItemText(0, \_translate("Dialog", "good"))

self.tasteinput.setItemText(1, \_translate("Dialog", "pure"))

self.pushButton.setText(\_translate("Dialog", "检测"))

self.Button.setText(\_translate("Dialog", "一键获取"))

self.Button2.setText(\_translate("Dialog", "返回"))

#预测结果可视化，对用户的数据定性+定量进行化分析

from PyQt5 import QtCore, QtGui, QtWidgets

from PyQt5.QtCore import QCoreApplication

from PyQt5.QtWidgets import QMessageBox

#疾病预测结果展示子界面GUI设计，大小按键设计

class Ui\_Dialog(object):

def setupUi(self, result1,result2,score1,score2,Dialog):

Dialog.setObjectName("Dialog")

Dialog.resize(1600, 1000)

Dialog.setFixedSize(1600, 1000)

self.label = QtWidgets.QLabel(Dialog)

self.label.setGeometry(QtCore.QRect(150, 240, 300, 40))

self.label.setObjectName("label")

self.label.setStyleSheet('font-size:35px')

self.label\_2 = QtWidgets.QLabel(Dialog)

self.label\_2.setGeometry(QtCore.QRect(420, 240, 100, 40))

self.label\_2.setObjectName("label\_2")

self.label\_2.setStyleSheet('font-size:35px')

self.label\_3 = QtWidgets.QLabel(Dialog)

self.label\_3.setGeometry(QtCore.QRect(100, 400, 537, 213))

self.label\_3.setObjectName("label\_3")

self.label\_4 = QtWidgets.QLabel(Dialog)

self.label\_4.setGeometry(QtCore.QRect(950, 240, 300, 40))

self.label\_4.setObjectName("label\_4")

self.label\_4.setStyleSheet('font-size:35px')

self.label\_5 = QtWidgets.QLabel(Dialog)

self.label\_5.setGeometry(QtCore.QRect(1220, 240, 100, 40))

self.label\_5.setObjectName("label\_5")

self.label\_5.setStyleSheet('font-size:35px')

self.label\_6 = QtWidgets.QLabel(Dialog)

self.label\_6.setGeometry(QtCore.QRect(900, 400, 537, 213))

self.label\_6.setObjectName("label\_6")

self.pushButton = QtWidgets.QPushButton(Dialog)

self.pushButton.setGeometry(QtCore.QRect(700, 800, 200, 40))

self.pushButton.setObjectName("pushButton")

self.pushButton.setStyleSheet('font-size:30px')

#需要用到的图片调用地址

self.png1 = QtGui.QPixmap('./databank/healthy1.png')

self.png2 = QtGui.QPixmap('./databank/healthy2.png')

self.png3 = QtGui.QPixmap('./databank/weak1.png')

self.png4 = QtGui.QPixmap('./databank/weak2.png')

self.result1 = result1

self.result2 = result2

self.score1 = score1

self.score2 = score2

self.retranslateUi(Dialog)

QtCore.QMetaObject.connectSlotsByName(Dialog)

#对疾病预测结果进行分析，是轻度或重度中毒

def retranslateUi(self, Dialog):

\_translate = QtCore.QCoreApplication.translate

Dialog.setWindowTitle(\_translate("Dialog", "预测结果"))

self.label.setText(\_translate("Dialog", "您心脏的状态是："))

self.label\_4.setText(\_translate("Dialog", "您肾脏的状态是："))

self.pushButton.setText(\_translate("Dialog", "退出"))

print(self.score1,self.score2)

if self.result1 == 0:

self.label\_2.setText(\_translate("Dialog", "健康"))

if self.score1 < 1:

self.label\_3.setPixmap(self.png1)

#print函数仅在编译时输出，用于检测程序debug

print("ok")

else :

self.label\_3.setPixmap(self.png2)

print("ok")

if self.result2 == 0:

self.label\_5.setText(\_translate("Dialog", "健康"))

if self.score2 < 1:

self.label\_6.setPixmap(self.png1)

print("ok")

else :

self.label\_6.setPixmap(self.png2)

print("ok")

if self.result1 == 1:

self.label\_2.setText(\_translate("Dialog", "虚弱"))

if self.score1 < 1:

self.label\_3.setPixmap(self.png3)

print("ok")

else :

self.label\_3.setPixmap(self.png4)

print("ok")

if self.result2 == 1:

self.label\_5.setText(\_translate("Dialog", "虚弱"))

if self.score2 < 1:

self.label\_6.setPixmap(self.png3)

print("ok")

else :

self.label\_6.setPixmap(self.png4)

print("ok")

#药物查询

#导入库函数

from PyQt5 import QtCore, QtGui, QtWidgets

from databank import medicineres

#GUI界面大小，按键设置

class Ui\_Dialog(object):

def setupUi(self, Dialog):

Dialog.setObjectName("Dialog")

Dialog.resize(1600, 1000)

Dialog.setFixedSize(1600, 1000)

self.label = QtWidgets.QLabel(Dialog)

self.label.setGeometry(QtCore.QRect(680, 60, 250, 61))

self.label.setObjectName("label")

self.label.setStyleSheet('font-size:40px')

self.label\_2 = QtWidgets.QLabel(Dialog)

self.label\_2.setGeometry(QtCore.QRect(640, 200, 450, 400))

self.label\_2.setObjectName("label\_2")

self.label\_2.setStyleSheet('font-size:25px')

self.pushButton\_3 = QtWidgets.QPushButton(Dialog)

self.pushButton\_3.setGeometry(QtCore.QRect(720, 800, 150, 50))

self.pushButton\_3.setObjectName("pushButton\_3")

self.pushButton\_3.setStyleSheet('font-size:30px')

self.pushButton\_3.clicked.connect(self.getresult)

self.textEdit = QtWidgets.QTextEdit(Dialog)

self.textEdit.setGeometry(QtCore.QRect(660, 500, 291, 201))

self.textEdit.setObjectName("textEdit")

self.textEdit.setStyleSheet('font-size:30px')

self.pushButton\_2 = QtWidgets.QPushButton(Dialog)

self.pushButton\_2.setGeometry(QtCore.QRect(720, 900, 150, 50))

self.pushButton\_2.setObjectName("pushButton\_3")

self.pushButton\_2.setStyleSheet('font-size:30px')

self.retranslateUi(Dialog)

QtCore.QMetaObject.connectSlotsByName(Dialog)

#接收用户输入，调用数据库

def getresult(self):

condition = self.textEdit.toPlainText()

self.a\_widget = QtWidgets.QWidget()

self.a = medicineres.Ui\_Dialog()

self.a.setupUi(self.a\_widget,condition)

self.a.pushButton.clicked.connect(self.a\_widget.close)

self.a\_widget.show()

#输出匹配到数据库中的数据

def retranslateUi(self, Dialog):

\_translate = QtCore.QCoreApplication.translate

Dialog.setWindowTitle(\_translate("Dialog", "Dialog"))

self.label.setText(\_translate("Dialog", "药物推荐助手"))

self.label\_2.setText(\_translate("Dialog","请输入您的症状以#作为间隔"))

self.pushButton\_3.setText(\_translate("Dialog", "查询"))

self.pushButton\_2.setText(\_translate("Dialog", "返回"))

#将匹配到的数据库结果可视化

#导入库函数

from PyQt5 import QtCore, QtGui, QtWidgets

import pandas as pd

#GUI界面设计，大小和按钮

class Ui\_Dialog(object):

def setupUi(self, Dialog,condition):

\_translate = QtCore.QCoreApplication.translate

Dialog.setObjectName("Dialog")

Dialog.setWindowTitle(\_translate("Dialog", "查询结果"))

Dialog.resize(600, 400)

Dialog.setFixedSize(600, 400)

self.pushButton = QtWidgets.QPushButton(Dialog)

self.pushButton.setGeometry(QtCore.QRect(200, 300, 200, 40))

self.pushButton.setObjectName("pushButton")

self.pushButton.setStyleSheet('font-size:30px')

self.pushButton.setText(\_translate("Dialog", "退出"))

self.tableWidget = QtWidgets.QTableWidget(Dialog)

self.tableWidget.setGeometry(QtCore.QRect(0, 0, 600, 300))

self.tableWidget.setObjectName("tableWidget") self.tableWidget.setEditTriggers(QtWidgets.QAbstractItemView.NoEditTriggers)

self.tableWidget.setColumnCount(3)

item = QtWidgets.QTableWidgetItem()

#接收输入的用户症状

item.setText(\_translate("Dialog", "medicine\_1"))

self.tableWidget.setHorizontalHeaderItem(0, item)

item = QtWidgets.QTableWidgetItem()

item.setText(\_translate("Dialog", "medicine\_2"))

self.tableWidget.setHorizontalHeaderItem(1, item)

item = QtWidgets.QTableWidgetItem()

item.setText(\_translate("Dialog", "medicine\_3"))

print(item)

self.tableWidget.setHorizontalHeaderItem(2, item)

self.condition = condition.split("#")

#读数据库

df = pd.read\_csv("./databank/cure\_clean.csv")

self.tableWidget.setRowCount(len(self.condition))

for i in self.condition :

medicine = []

medicine\_1 = df.loc[df['condition'] == i, 'medicine\_1']

medicine\_2 = df.loc[df['condition'] == i, 'medicine\_2']

medicine\_3 = df.loc[df['condition'] == i, 'medicine\_3']

if len(medicine\_1) == 0 or len(medicine\_2) == 0 or len(medicine\_3) == 0 :

info = " %s Not Found" % (i)

QtWidgets.QMessageBox.about(None,"Warning",info) self.tableWidget.setVerticalHeaderItem(self.condition.index(i), QtWidgets.QTableWidgetItem(i))

else : self.tableWidget.setVerticalHeaderItem(self.condition.index(i), QtWidgets.QTableWidgetItem(i))

#输出排名前三的药物

medicine\_1 = medicine\_1.values[0]

medicine\_2 = medicine\_2.values[0]

medicine\_3 = medicine\_3.values[0]

medicine\_1= medicine\_1.strip().replace('Series([], )', ' ')

medicine\_2= medicine\_2.strip().replace('Series([], )', ' ')

medicine\_3= medicine\_3.strip().replace('Series([], )', ' ')

medicine.append(medicine\_1)

medicine.append(medicine\_2)

medicine.append(medicine\_3)

for j in medicine:

self.tableWidget.setItem(self.condition.index(i), medicine.index(j),

QtWidgets.QTableWidgetItem(j))

QtWidgets.QTableWidget.resizeColumnsToContents(self.tableWidget)

QtCore.QMetaObject.connectSlotsByName(Dialog)

#测试文件代码

#导入库函数

import PyQt5

from PyQt5 import QtCore, QtGui, QtWidgets

from PyQt5.QtCore import QCoreApplication

import sys,xlsxwriter,csv,os

from databank import jiance,mainwindow,medicine,medicineres

#打开疾病预测模块，预测疾病；创建或打开疾病记录，记录用户数据

with open("./databank/test.csv", 'w') as f:

csv\_write = csv.writer(f)

data\_row = ["age", "sex","trestbps","chol","fbs","restecg","thalach","exang","oldpeak","ca","cp\_0","cp\_1","cp\_2","cp\_3","thal\_0","thal\_1","thal\_2","thal\_3","slope\_0","slope\_1","slope\_2"]

csv\_write.writerow(data\_row)

f.close()

with open("./databank/test2.csv", 'w') as f:

csv\_write = csv.writer(f)

data\_row = ["age", "bp","sg","al","su","rbc","pc","pcc","ba","bgr","bu","sc","sod","pot","hemo","htn","dm","cad","appet","pe","ane"]

csv\_write.writerow(data\_row)

f.close()

#打开药物推荐模块，推荐药物

app = QtWidgets.QApplication(sys.argv)

a\_widget = QtWidgets.QWidget()

b\_widget = QtWidgets.QWidget()

c\_widget = QtWidgets.QWidget()

#界面GUI设计及输出

a = mainwindow.Ui\_Dialog()

a.setupUi(a\_widget)

a\_widget.show()

b = jiance.Ui\_Dialog()

b.setupUi(b\_widget)

c = medicine.Ui\_Dialog()

c.setupUi(c\_widget)

a.pushButton.clicked.connect(b\_widget.show)

c.pushButton\_2.clicked.connect(c\_widget.close)

b.Button2.clicked.connect(b\_widget.close)

a.pushButton\_2.clicked.connect(c\_widget.show)

sys.exit(app.exec\_())

项目3 基于SVM的酒店评论推荐系统

3.3 模块实现

3.3.1 数据预处理

1.数据整合

#读取每一条文字内容

def getContent(fullname):

f = open(fullname,'rb+')

content = f.readlines()

f.close()

return content

#将积极和消极评论分别写入两个文件中

for parent,dirnames,filenames in os.walk(rootdir):

for filename in filenames:

#使用getContent()函数，得到每条评论的具体内容

content = getContent(rootdir + '\\' + filename)

output.writelines(content)

i = i+1

output.close()

2.文本清洗

#文本清洗

def clearTxt(line):

if line != '':

#去掉末尾的空格

line = line.strip()

pun\_num = string.punctuation + string.digits

intab = pun\_num

outtab = " "\*len(pun\_num)

#去除所有标点和数字

trantab = str.maketrans(intab, outtab)

line = line.translate(trantab)

#去除文本中的英文和数字

line = re.sub("[a-zA-Z0-9]", "", line)

#去除文本中的中文符号和英文符号

line = re.sub("[\s+\.\!\/\_,$%^\*(+\"\'；：“”．]+|[+——！=＝°【】，÷。？?、 ~@#￥%……&\*（）]+", "", line)

return line

#进行文本分词

#引入jieba模块

import jieba

import jieba.analyse

import codecs,sys,string,re

#文本分词

def sent2word(line):

segList = jieba.cut(line,cut\_all=False)

segSentence = ''

for word in segList:

if word != '\t':

segSentence += word + " "

return segSentence.strip()

#删除分词后文本里的停用词

def delstopword(line,stopkey):

wordList = line.split(' ')

sentence = ''

for word in wordList:

word = word.strip()

#spotkey是在主函数中获取的评论行数

#逐行删除，不破坏词所在每行的位置，始终保持每条评论的间隔

if word not in stopkey:

if word != '\t':

sentence += word + " "

return sentence.strip()

3.文本分词

#载入模型

fdir = 'E:\word2vec\word2vec\_from\_weixin\word2vec'

inp = fdir + '\word2vec\_wx'

model = gensim.models.Word2Vec.load(inp)

#把词语转化为词向量的函数

def getWordVecs(wordList,model):

vecs = []

for word in wordList:

word = word.replace('\n','')

#print word

try:

vecs.append(model[word])

except KeyError:

continue

return np.array(vecs, dtype='float')

#转化为词向量

def buildVecs(filename,model):

fileVecs = []

with codecs.open(filename, 'rb', encoding='utf-8') as contents:

for line in contents:

wordList = line.split(' ')

#调用getwordVecs()函数，获取每条评论的词向量

vecs = getWordVecs(wordList,model)

if len(vecs) >0:

vecsArray = sum(np.array(vecs))/len(vecs)

fileVecs.append(vecsArray)

return fileVecs

#建立词向量表，其中积极的首列填充为1，消极的首列填充为0

Y = np.concatenate((np.ones(len(posInput)), np.zeros(len(negInput))))

X = posInput[:]

for neg in negInput:

X.append(neg)

X = np.array(X)

3.3.2 模型训练及保存

1.加载词向量表，并设置训练集和测试集

相关代码如下：

fdir = ''

df = pd.read\_csv(fdir + '2000\_data.csv')

#导入每一条词向量为x，对应的结果为y

y = df.iloc[:,1].values

x = df.iloc[:,2:].values

#分割训练集和测试集

(x\_train,x\_test,y\_train,y\_test)=train\_test\_split(x,y,test\_size=0.2,random\_state=1)

2.模型训练并保存

相关代码如下：

#进行模型的训练

clf = svm.SVC(C = 10,kernel='rbf',gamma=0.38,probability = True)#训练

clf.fit(x\_train,y\_train)

#打印模型在训练集上的准确率

print 'train Accuracy: %.2f'% clf.score(x\_train,y\_train)

#打印模型在测试集上的准确率

print 'Test Accuracy: %.2f'% clf.score(x\_test,y\_test)

pred\_probas = clf.predict\_proba(x)[:,1] #score

fpr,tpr,\_ = metrics.roc\_curve(y, pred\_probas)

roc\_auc = metrics.auc(fpr,tpr)

#画出模型的ROC曲线，便于后续分析调整模型

plt.plot(fpr, tpr, label = 'area = %.2f' % roc\_auc)

plt.plot([0, 1], [0, 1], 'k--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.legend(loc = 'lower right')

plt.show()

#保存模型

joblib.dump(clf, "my\_model\_1.m")

3.3.3 模型应用

1.爬取评论

在携程酒店爬取指定酒店ID的评论集。

#爬虫输入网页请求参数，得到相应网页

def getResponse(url,pageindex):

data={"hotelId":1737627,"pageIndex":pageindex,"tagId":0,"pageSize":10,"groupTypeBitMap":2,"needStatisticInfo":0,"order":0,"basicRoomName":"","travelType":1,"head":{"cid":"09031179411625216472","ctok":"","cver":"1.0","lang":"01","sid":"8888","syscode":"09","auth":"","extension":[]}}

data = json.dumps(data).encode(encoding='utf-8')

#模拟普通浏览器的方式

header\_dict = {'User-Agent': 'Mozilla/5.0 (Windows NT 6.1; Trident/7.0; rv:11.0) like Gecko',"Content-Type": "application/json"}

url\_request = request.Request(url=url,data=data,headers=header\_dict)

print("正在采集的是第%d页", %i)

url\_response = request.urlopen(url\_request)

return url\_response

if \_\_name\_\_ == "\_\_main\_\_":

#循环爬取所有页数

for i in range(1,434):

#输入采取网页的地址

http\_response=getResponse("http://m.ctrip.com/restapi/soa2/16765/gethotelcomment?\_fxpcqlniredt=09031144211504567945",i)

data = http\_response.read().decode('utf-8')

dic = json.loads(data)

ungz=dic['othersCommentList']

#网页结构是每页有十条评论

for k in range(10):

content=ungz[k]

comment=content['content']

#存储到本地文件中

with open('comment\_beiwai.txt', 'a', encoding='UTF-8') as f:

f.write(json.dumps(comment, ensure\_ascii=False) + '\n'+'\n')

#每采集页延迟10s

time.sleep(10)

#将爬取的评论写入数据库中，使用longtext存储，为节省空间只存储了前一百个字符

def into(path,string):

#连接数据库

conn = mysql.connector.connect(user='root', password='password', database='grades', use\_unicode=True)

cursor = conn.cursor()

f = codecs.open(path, 'r', encoding='utf-8')

f=f.read()

#删去评论里的换行符，节省存储空间

f=filter\_emoji(f,restr='')

f=f.replace('\r','')

f=f.replace('\n','')

f=f.replace('"',"")

f=f[0:100]

print(f)

def filter\_emoji(desstr,restr=""):

#过滤表情

try:

co = re.compile(u'[\U00010000-\U0010ffff]')

except re.error:

co = re.compile(u'[\uD800-\uDBFF][\uDC00-\uDFFF]')

return co.sub(restr, desstr)

try:

print(string)

print(f)

#插入酒店和分数到数据库

cursor.execute('insert into grades (hotel,date) values(%s,%s)',[string,f]) conn.commit()

except:

#插入不成功回滚，并且报错

conn.rollback()

print("fail")

2.酒店打分

import warnings

warnings.filterwarnings(action='ignore', category=UserWarning, module='gensim') #忽略警告

import logging

import os.path

import codecs,sys

import numpy as np

import pandas as pd

import gensim

import matplotlib.pyplot as plt

from sklearn.decomposition import PCA

from sklearn import svm

from sklearn import metrics

from sklearn.externals import joblib

import collections

import mysql.connector

import jieba

import jieba.analyse

import string,re

def prepareData(sourceFile,targetFile):

f = open(sourceFile, 'r', encoding='utf-8')

target = ""

print( 'open source file: '+ sourceFile)

print ('open target file: '+ targetFile)

lineNum = 1

line = f.readline()

while line:

print( '---processing ',lineNum,' article---')

line = clearTxt(line)

seg\_line = sent2word(line)

target.writelines(seg\_line + '\n')

lineNum = lineNum + 1

line = f.readline()

f.close()

return target

#清洗文本

def clearTxt(line):

if line != '':

line = line.strip()

pun\_num = string.punctuation + string.digits

intab = pun\_num

outtab = " "\*len(pun\_num)

#去除所有标点和数字

trantab = str.maketrans(intab, outtab)

line = line.translate(trantab)

#去除文本中的英文和数字

line = re.sub("[a-zA-Z0-9]", "", line)

#去除文本中的中文符号和英文符号

line = re.sub("[\s+\.\!\/\_,$%^\*(+\"\'；：“”．]+|[+——！=＝°【】，÷。？?、~@#￥%……&\*（）]+", "", line)

return line

#文本切割

def sent2word(line):

segList = jieba.cut(line,cut\_all=False)

segSentence = ''

for word in segList:

if word != '\t':

segSentence += word + " "

return segSentence.strip()

def stopWord(source,stopkey):

sourcef = source

lineNum = 1

line = sourcef.readline()

target=""

while line:

print ('---processing ',lineNum,' article---')

sentence = delstopword(line,stopkey)

#print sentence

target.writelines(sentence + '\n')

lineNum = lineNum + 1

line = sourcef.readline()

return target

#构建特征词向量

def getWordVecs(wordList,model):

vecs = []

for word in wordList:

word = word.replace('\n','')

#print word

try:

vecs.append(model[word])

except KeyError:

continue

return np.array(vecs, dtype='float')

#构建文档词向量

def buildVecs(input,model):

fileVecs = []

with codecs.open(filename, 'rb', encoding='utf-8') as contents:

for line in contents:

logger.info("Start line: " + line)

wordList = line.split(' ')

vecs = getWordVecs(wordList,model)

#print vecs

#sys.exit()

#for each sentence, the mean vector of all its vectors is used to represent this sentence

if len(vecs) >0:

vecsArray = sum(np.array(vecs))/len(vecs) # mean

#print vecsArray

#sys.exit()

fileVecs.append(vecsArray)

return fileVecs

if \_\_name\_\_ == '\_\_main\_\_':

sourceFile = '../comment.txt'

#分词

result1=prepareData(sourceFile)

stopkey = [w.strip() for w in result1.readlines()]

#去除停用词

result2=stopWord(source,stopkey)

program = os.path.basename(sys.argv[0])

logger = logging.getLogger(program) logging.basicConfig(format='%(asctime)s: %(levelname)s: %(message)s',level=logging.INFO)

logger.info("running %s" % ' '.join(sys.argv))

#加载模型

fdir = 'E:\word2vec\word2vec\_from\_weixin\word2vec'

inp = fdir + '\word2vec\_wx'

model = gensim.models.Word2Vec.load(inp)

fdir1 = 'C:\Users\zhoua\\test\_a\_review'

#开始构建词向量表

output = buildVecs(result2,model)

Y = np.ones(len(posInput))

X = output[:]

X = np.array(X)

df\_x = pd.DataFrame(X)

df\_y = pd.DataFrame(Y)

data = pd.concat([df\_y,df\_x],axis = 1)

data.to\_csv(fdir1 + '\\comment\_data.csv')

#加载模型，统计模型的0与1占比，根据一定规则进行打分，并且将分数存入数据库

clf = joblib.load("my\_model\_1.m")

def countgrades(filename):

df = pd.read\_csv(filename)

x = df.iloc[:,2:]

predict=clf.predict(x)

pred\_probas = clf.predict\_proba(x)

x=collections.Counter(predict)

#统计0和1 的比例，并给出分数

x0=x[0]

x1=x[1]

grades=100\*x1/(x1+x0)

return grades

if \_\_name\_\_=="\_\_main\_\_":

#存入数据库

conn = mysql.connector.connect(user='root', password='zhou19990806', database='grades', use\_unicode=True)

cursor = conn.cursor()

filename1="comment\_data.csv"

grades1=countgrades(filename1)

cursor.execute('insert into grades grades=%s where hotel=%s', (int(grades1),'南京全季酒店'))

print '南京全季酒店的分数是: %.2f'% grades1

filename2="comment\_bupt\_data.csv"

grades2=countgrades(filename2)

cursor.execute('insert into grades grades=%s where hotel=%s', (int(grades2),'北邮科技酒店'))

print '北邮科技酒店的分数是: %.2f'% grades2

filename3="comment\_pku\_data.csv"

grades3=countgrades(filename3)

cursor.execute('insert into grades grades=%s where hotel=%s', (int(grades3),'北大博雅酒店'))

print '北大博雅酒店的分数是: %.2f'% grades3

filename4="comment\_beiwai\_data.csv"

grades4=countgrades(filename4)

cursor.execute('insert into grades grades=%s where hotel=%s', (int(grades4),'鹤佳酒店'))

print '鹤佳酒店的分数是: %.2f'% grades4

filename5="comment\_beijiao\_data.csv"

grades5=countgrades(filename5)

cursor.execute('insert into grades grades=%s where hotel=%s', (int(grades5),'乐家服务酒店'))

print '乐家服务酒店的分数是: %.2f'% grades5

conn.commit()

cursor.close()

3.界面设置

##### 1)创建Django项目

进入命令行环境，输入命令，生成helloworld项目：

django-admin startproject HelloWorld

##### 2)html布局文件

相关代码如下：

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>酒店分数查询</title>

<style type="text/css">

<!—表单的样式-->

#bg{

width:700px;

height:400px;

margin:0 auto;

}

.bordered {

border-style:solid;

width:600px;

height:300px;

}

</style>

</head>

<body>

<!—查询框-->

<body align="center"/>

<body style="color:black"/>

<br/><br/><br/><br/> &nbsp;&nbsp;&nbsp;&nbsp;

<div id="bg" class="bordered">

<fieldset>

<legend><h1 align="center" >查询页面</h1></legend>

<form action="/search-post" method="post">

{% csrf\_token %}

<p> <strong>酒店名</strong>&nbsp;<input type="text" name="q"></p>

<br/>

<p><input type="submit"value="查询"style="width:70px;height:30px" ></p>

</form>

<!—预留内容，展示输出内容-->

</fieldset>

<p id="view">{{ rlt1 }}</p>

<p id="view">{{ rlt2 }}</p>

<div>

</body>

</html>

##### 3)后台调用数据库

相关代码如下：

import mysql.connector

#接收POST请求数据

def search\_post(request):

#连接数据库

conn = mysql.connector.connect(user='root', password='password', database='grades', use\_unicode=True)

cursor = conn.cursor()

ctx ={}

if request.POST:

#获取表单中填入内容

res=request.POST['q']

#在数据库中搜索并展示结果

sql="SELECT \* FROM grades WHERE hotel=%s"

cursor.execute(sql,(res,))

results = cursor.fetchall()

results=results[0]

grade=results[1]

data=results[2]

#展示该酒店的打分

ctx['rlt1'] = "分数："+str(grade)

#展示该酒店的评论

ctx['rlt2']="评论："+str(data)+"......"

cursor.close()

#返回响应内容

return render(request, "hello.html", ctx)

##### 4)Django项目中其余两个重要文件

相关代码如下：

#Django中自动生成的urls.py文件，将提交动作与后台响应函数绑定

from django.conf.urls import url

from . import view

urlpatterns = [

url(r'^search-post$', view.search\_post),

]

#Django中自动生成的settings.py文件

import os

BASE\_DIR = os.path.dirname(os.path.dirname(os.path.abspath(\_\_file\_\_)))

SECRET\_KEY = '\_$&5uv^$+5@$\*&&%9c+0+-c7v8%dmsj(ycnq=sh34a\_)s+7n=p'

DEBUG = True

ALLOWED\_HOSTS = ['\*']

INSTALLED\_APPS = [

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

]

MIDDLEWARE = [

'django.middleware.security.SecurityMiddleware',

'django.contrib.sessions.middleware.SessionMiddleware',

'django.middleware.common.CommonMiddleware',

'django.middleware.csrf.CsrfViewMiddleware',

'django.contrib.auth.middleware.AuthenticationMiddleware',

'django.contrib.messages.middleware.MessageMiddleware',

'django.middleware.clickjacking.XFrameOptionsMiddleware',

]

ROOT\_URLCONF = 'HelloWorld.urls'

TEMPLATES = [

{

'BACKEND': 'django.template.backends.django.DjangoTemplates',

'DIRS': [BASE\_DIR+"/templates",],

'APP\_DIRS': True,

'OPTIONS': {

'context\_processors': [

'django.template.context\_processors.debug',

'django.template.context\_processors.request',

'django.contrib.auth.context\_processors.auth',

'django.contrib.messages.context\_processors.messages',

],

},

},

]

WSGI\_APPLICATION = 'HelloWorld.wsgi.application'

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.sqlite3',

'NAME': os.path.join(BASE\_DIR, 'db.sqlite3'),

}

}

AUTH\_PASSWORD\_VALIDATORS = [

{

'NAME': 'django.contrib.auth.password\_validation.UserAttributeSimilarityValidator',

},

{

'NAME': 'django.contrib.auth.password\_validation.MinimumLengthValidator',

},

{

'NAME': 'django.contrib.auth.password\_validation.CommonPasswordValidator',

},

{

'NAME': 'django.contrib.auth.password\_validation.NumericPasswordValidator',

},

]

LANGUAGE\_CODE = 'en-us'

TIME\_ZONE = 'UTC'

USE\_I18N = True

USE\_L10N = True

USE\_TZ = True

STATIC\_URL = '/static/'

项目4 基于MovieLens数据集的电影推荐系统

4.3 模块实现

4.3.1 模型训练

1.数据集分析

相关代码如下:

ratings\_title = ['UserID','MovieID', 'Rating', 'timestamps']

ratings = pd.read\_table('./ml-1m/ratings.dat', sep='::', header=None, names=ratings\_title, engine = 'python')

ratings.head()

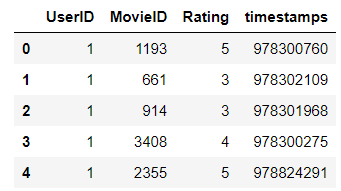


图4-7 ratings.dat数据

使用MovieID、Title和Genres，MovieID和Genres是类别字段，Title是文本。Title与IMDB提供的标题相同（包括发行年份），Genres是管道分隔，并且选自以下流派：

Action Adventure Animation Children's Comedy

Crime Documentary Drama Fantasy Film-Noir

Horror Musical Mystery Romance Sci-Fi

Thriller War Western

查看movies.dat中前5个数据，结果如图4-8所示，相关代码如下：

movies\_title = ['MovieID', 'Title', 'Genres']

movies = pd.read\_table('./ml-1m/movies.dat', sep='::', header=None, names = movies\_title, engine = 'python')

movies.head()

2.数据预处理

相关代码如下：

#数据预处理

def load\_data():

#处理 users.dat

users\_title = ['UserID', 'Gender', 'Age', 'JobID', 'Zip-code']

users = pd.read\_table('./ml-1m/users.dat', sep='::', header=None, names=users\_title, engine = 'python')

#去除邮编

users = users.filter(regex='UserID|Gender|Age|JobID')

users\_orig = users.values

#改变数据中的性别和年龄

gender\_map = {'F':0, 'M':1}

users['Gender'] = users['Gender'].map(gender\_map)

age\_map = {val:ii for ii,val in enumerate(set(users['Age']))}

users['Age'] = users['Age'].map(age\_map)

#处理 movies.dat

movies\_title = ['MovieID', 'Title', 'Genres']

movies = pd.read\_table('./ml-1m/movies.dat', sep='::', header=None, names=movies\_title, engine = 'python')

movies\_orig = movies.values

#去掉Title中的年份

pattern = re.compile(r'^(.\*)\((\d+)\)$')

title\_map = {val:pattern.match(val).group(1) for ii,val in enumerate(set(movies['Title']))}

movies['Title'] = movies['Title'].map(title\_map)

#电影类型转数字字典

genres\_set = set()

for val in movies['Genres'].str.split('|'):

genres\_set.update(val)

genres\_set.add('<PAD>')

genres2int = {val:ii for ii, val in enumerate(genres\_set)}

#将电影类型转成等长数字列表，长度是18

genres\_map = {val:[genres2int[row] for row in val.split('|')] for ii,val in enumerate(set(movies['Genres']))}

for key in genres\_map:

for cnt in range(max(genres2int.values()) - len(genres\_map[key])):

genres\_map[key].insert(len(genres\_map[key])+ cnt,genres2int['<PAD>'])

movies['Genres'] = movies['Genres'].map(genres\_map)

#电影Title转数字字典

title\_set = set()

for val in movies['Title'].str.split():

title\_set.update(val)

title\_set.add('<PAD>')

title2int = {val:ii for ii, val in enumerate(title\_set)}

#将电影Title转成等长数字列表，长度是15

title\_count = 15

title\_map = {val:[title2int[row] for row in val.split()] for ii,val in enumerate(set(movies['Title']))}

for key in title\_map:

for cnt in range(title\_count - len(title\_map[key])):

title\_map[key].insert(len(title\_map[key]) + cnt,title2int['<PAD>'])

movies['Title'] = movies['Title'].map(title\_map)

#处理 ratings.dat

ratings\_title = ['UserID','MovieID', 'ratings', 'timestamps']

ratings = pd.read\_table('./ml-1m/ratings.dat', sep='::', header=None, names=ratings\_title, engine = 'python')

ratings = ratings.filter(regex='UserID|MovieID|ratings')

#合并三个表

data = pd.merge(pd.merge(ratings, users), movies)

#将数据分成X和y两张表

target\_fields = ['ratings']

features\_pd, targets\_pd = data.drop(target\_fields, axis=1), data[target\_fields]

features = features\_pd.values

targets\_values = targets\_pd.values

return title\_count, title\_set, genres2int, features, targets\_values, ratings, users, movies, data, movies\_orig, users\_orig

#加载数据并保存到本地

#title\_count：Title字段的长度（15）

#title\_set：Title文本的集合

#genres2int：电影类型转数字的字典

#features：是输入X

#targets\_values：是学习目标y

#ratings：评分数据集的Pandas对象

#users：用户数据集的Pandas对象

#movies：电影数据的Pandas对象

#data：三个数据集组合在一起的Pandas对象

#movies\_orig：没有做数据处理的原始电影数据

#users\_orig：没有做数据处理的原始用户数据

#调用数据处理函数

title\_count, title\_set, genres2int, features, targets\_values, ratings, users, movies, data, movies\_orig, users\_orig = load\_data()

#保存预处理结果

pickle.dump((title\_count, title\_set, genres2int, features,

targets\_values, ratings, users, movies, data,

movies\_orig, users\_orig), open('preprocess.p', 'wb'))

查看预处理后的数据，如图4-9所示。

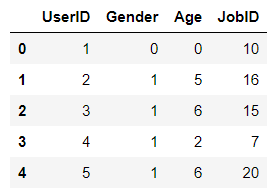


图4-9 预处理后users数据

处理后的movies数据如图4-10所示。

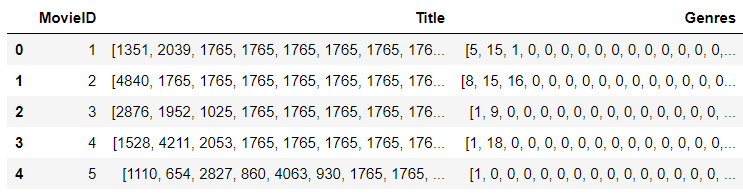


图4-10 预处理后的movies数据

3.模型创建

相关代码如下：

title\_count, title\_set, genres2int, features, targets\_values, ratings, users, movies, data, movies\_orig, users\_orig = pickle.load(open('preprocess.p', mode='rb'))

加载数据后定义神经网络的模型结构：

1)定义参数

相关代码如下：

#嵌入矩阵的维度

embed\_dim = 32

#用户ID个数

uid\_max = max(features.take(0,1)) + 1 #6040

#性别个数

gender\_max = max(features.take(2,1)) + 1 #1+1=2

#年龄类别个数

age\_max = max(features.take(3,1)) + 1 #6+1=7

#职业个数

job\_max = max(features.take(4,1)) + 1 #20+1=21

#电影ID个数

movie\_id\_max = max(features.take(1,1)) + 1 #3952

#电影类型个数

movie\_categories\_max = max(genres2int.values()) + 1 #18+1=19

#电影名单词个数

movie\_title\_max = len(title\_set) #5216

#对电影类型嵌入向量做加和操作的标志，考虑过使用mean做平均，但是没实现mean

combiner = "sum"

#电影名长度

sentences\_size = title\_count #15

#文本卷积滑动窗口，分别滑动2,3, 4, 5个单词

window\_sizes = {2, 3, 4, 5}

#文本卷积核数量

filter\_num = 8

#电影ID转下标的字典，数据集中电影ID跟下标不一致，例如第5行的数据电影ID不一定是5

movieid2idx = {val[0]:i for i, val in enumerate(movies.values)}

2)定义网络数据输入占位符

def get\_inputs():

#输入占位符

#用户数据输入

uid = tf.placeholder(tf.int32, [None, 1], name="uid")

user\_gender = tf.placeholder(tf.int32, [None, 1], name="user\_gender")

user\_age = tf.placeholder(tf.int32, [None, 1], name="user\_age")

user\_job = tf.placeholder(tf.int32, [None, 1], name="user\_job")

#电影数据输入

movie\_id = tf.placeholder(tf.int32, [None, 1], name="movie\_id")

movie\_categories = tf.placeholder(tf.int32, [None, 18], name="movie\_categories")

movie\_titles = tf.placeholder(tf.int32, [None, 15], name="movie\_titles")

#目标评分

targets = tf.placeholder(tf.int32, [None, 1], name="targets")

#学习率

LearningRate = tf.placeholder(tf.float32, name = "LearningRate")

#弃用率

dropout\_keep\_prob = tf.placeholder(tf.float32, name = "dropout\_keep\_prob")

return uid, user\_gender, user\_age, user\_job, movie\_id, movie\_categories, movie\_titles, targets, LearningRate, dropout\_keep\_prob

3)定义用户嵌入矩阵

相关代码如下：

def get\_user\_embedding(uid, user\_gender, user\_age, user\_job):

#定义User的嵌入矩阵，返回某个userid的嵌入层向量

with tf.name\_scope("user\_embedding"):

#用户ID嵌入矩阵

uid\_embed\_matrix = tf.Variable(tf.random\_uniform([uid\_max, embed\_dim], -1, 1), name = "uid\_embed\_matrix")

uid\_embed\_layer = tf.nn.embedding\_lookup(uid\_embed\_matrix, uid, name = "uid\_embed\_layer")

#用户性别嵌入矩阵

gender\_embed\_matrix = tf.Variable(tf.random\_uniform([gender\_max, embed\_dim // 2], -1, 1), name= "gender\_embed\_matrix")

gender\_embed\_layer = tf.nn.embedding\_lookup(gender\_embed\_matrix, user\_gender, name = "gender\_embed\_layer")

#用户年龄嵌入矩阵

age\_embed\_matrix = tf.Variable(tf.random\_uniform([age\_max, embed\_dim // 2], -1, 1), name="age\_embed\_matrix")

age\_embed\_layer = tf.nn.embedding\_lookup(age\_embed\_matrix, user\_age, name="age\_embed\_layer")

#用户职业嵌入矩阵

job\_embed\_matrix = tf.Variable(tf.random\_uniform([job\_max, embed\_dim // 2], -1, 1), name = "job\_embed\_matrix")

job\_embed\_layer = tf.nn.embedding\_lookup(job\_embed\_matrix, user\_job, name = "job\_embed\_layer")

return uid\_embed\_layer, gender\_embed\_layer, age\_embed\_layer, job\_embed\_layer

4)定义电影嵌入矩阵

相关代码如下：

def get\_movie\_id\_embed\_layer(movie\_id):

#定义MovieId的嵌入矩阵，返回某个电影ID的嵌入层向量

with tf.name\_scope("movie\_embedding"):

movie\_id\_embed\_matrix = tf.Variable(tf.random\_uniform([movie\_id\_max, embed\_dim], -1, 1), name = "movie\_id\_embed\_matrix")

movie\_id\_embed\_layer = tf.nn.embedding\_lookup(movie\_id\_embed\_matrix, movie\_id, name = "movie\_id\_embed\_layer")

return movie\_id\_embed\_layer

5)定义电影类型嵌入矩阵

相关代码如下：

def get\_movie\_categories\_layers(movie\_categories):

#电影类型的嵌入矩阵，返回某部电影所有类型向量的和

with tf.name\_scope("movie\_categories\_layers"):

#定义嵌入矩阵

movie\_categories\_embed\_matrix = tf.Variable(tf.random\_uniform([movie\_categories\_max, embed\_dim], -1, 1), name = "movie\_categories\_embed\_matrix")

#根据索引选择电影类型向量

movie\_categories\_embed\_layer = tf.nn.embedding\_lookup(movie\_categories\_embed\_matrix, movie\_categories, name = "movie\_categories\_embed\_layer")

#向量元素相加

if combiner == "sum":

movie\_categories\_embed\_layer = tf.reduce\_sum(movie\_categories\_embed\_layer, axis=1, keep\_dims=True)

return movie\_categories\_embed\_layer

6)处理电影名称

相关代码如下：

def get\_movie\_cnn\_layer(movie\_titles):

#从嵌入矩阵中得到电影名对应的各单词嵌入向量

with tf.name\_scope("movie\_embedding"): movie\_title\_embed\_matrix=tf.Variable(tf.random\_uniform([movie\_title\_max, embed\_dim], -1, 1), name = "movie\_title\_embed\_matrix") movie\_title\_embed\_layer=tf.nn.embedding\_lookup(movie\_title\_embed\_matrix, movie\_titles, name = "movie\_title\_embed\_layer")

movie\_title\_embed\_layer\_expand = tf.expand\_dims(movie\_title\_embed\_layer, -1)

#对文本嵌入层使用不同尺寸的卷积核做卷积和最大池化

pool\_layer\_lst = []

for window\_size in window\_sizes:

with tf.name\_scope("movie\_txt\_conv\_maxpool\_{}".format(window\_size)):

#权重

filter\_weights = tf.Variable(tf.truncated\_normal([window\_size, embed\_dim, 1, filter\_num],stddev=0.1),name = "filter\_weights")

filter\_bias = tf.Variable(tf.constant(0.1, shape=[filter\_num]), name="filter\_bias")

#卷积层

conv\_layer = tf.nn.conv2d(movie\_title\_embed\_layer\_expand, filter\_weights, [1,1,1,1], padding="VALID", name="conv\_layer")

relu\_layer = tf.nn.relu(tf.nn.bias\_add(conv\_layer,filter\_bias), name ="relu\_layer")

#池化层

maxpool\_layer = tf.nn.max\_pool(relu\_layer, [1,sentences\_size - window\_size + 1 ,1,1], [1,1,1,1], padding="VALID", name="maxpool\_layer")

pool\_layer\_lst.append(maxpool\_layer)

#丢弃层

with tf.name\_scope("pool\_dropout"):

pool\_layer = tf.concat(pool\_layer\_lst, 3, name ="pool\_layer")

max\_num = len(window\_sizes) \* filter\_num

pool\_layer\_flat = tf.reshape(pool\_layer , [-1, 1, max\_num], name = "pool\_layer\_flat")

dropout\_layer = tf.nn.dropout(pool\_layer\_flat, dropout\_keep\_prob, name = "dropout\_layer")

return pool\_layer\_flat, dropout\_layer

#用户嵌入层向量全连接层定义函数

def get\_user\_feature\_layer(uid\_embed\_layer, gender\_embed\_layer, age\_embed\_layer, job\_embed\_layer):

#用户嵌入层向量全连接

with tf.name\_scope("user\_fc"):

#第一层全连接

uid\_fc\_layer = tf.layers.dense(uid\_embed\_layer, embed\_dim, name = "uid\_fc\_layer", activation=tf.nn.relu)

gender\_fc\_layer = tf.layers.dense(gender\_embed\_layer, embed\_dim, name = "gender\_fc\_layer", activation=tf.nn.relu)

age\_fc\_layer = tf.layers.dense(age\_embed\_layer, embed\_dim, name ="age\_fc\_layer", activation=tf.nn.relu)

job\_fc\_layer = tf.layers.dense(job\_embed\_layer, embed\_dim, name = "job\_fc\_layer", activation=tf.nn.relu)

#第二层全连接

user\_combine\_layer = tf.concat([uid\_fc\_layer, gender\_fc\_layer, age\_fc\_layer, job\_fc\_layer], 2) #(?, 1, 128)

user\_combine\_layer = tf.contrib.layers.fully\_connected(user\_combine\_layer, 200, tf.tanh) #(?, 1, 200)

user\_combine\_layer\_flat = tf.reshape(user\_combine\_layer, [-1, 200])

return user\_combine\_layer, user\_combine\_layer\_flat

#电影特征全连接层定义函数

def get\_movie\_feature\_layer(movie\_id\_embed\_layer, movie\_categories\_embed\_layer, dropout\_layer):

#所有电影特征全连接

with tf.name\_scope("movie\_fc"):

#第一层全连接

movie\_id\_fc\_layer = tf.layers.dense(movie\_id\_embed\_layer, embed\_dim, name = "movie\_id\_fc\_layer", activation=tf.nn.relu)

movie\_categories\_fc\_layer = tf.layers.dense(movie\_categories\_embed\_layer, embed\_dim, name = "movie\_categories\_fc\_layer", activation=tf.nn.relu)

#第二层全连接

movie\_combine\_layer = tf.concat([movie\_id\_fc\_layer, movie\_categories\_fc\_layer, dropout\_layer], 2) #(?, 1, 96) movie\_combine\_layer=tf.contrib.layers.fully\_connected(movie\_combine\_layer,200,tf.tanh)#(?,1, 200)

movie\_combine\_layer\_flat = tf.reshape(movie\_combine\_layer, [-1, 200])

return movie\_combine\_layer, movie\_combine\_layer\_flat

8)定义计算图

相关代码如下：

#计算图定义

tf.reset\_default\_graph()

train\_graph = tf.Graph()

with train\_graph.as\_default():

#获取输入占位符

uid, user\_gender, user\_age, user\_job, movie\_id, movie\_categories, movie\_titles, targets, lr, dropout\_keep\_prob = get\_inputs()

#获取User的4个嵌入向量

uid\_embed\_layer, gender\_embed\_layer, age\_embed\_layer, job\_embed\_layer = get\_user\_embedding(uid, user\_gender, user\_age, user\_job)

#得到用户特征

user\_combine\_layer, user\_combine\_layer\_flat = get\_user\_feature\_layer(uid\_embed\_layer, gender\_embed\_layer, age\_embed\_layer, job\_embed\_layer)

#获取电影ID的嵌入向量

movie\_id\_embed\_layer = get\_movie\_id\_embed\_layer(movie\_id)

#获取电影类型的嵌入向量 movie\_categories\_embed\_layer=get\_movie\_categories\_layers(movie\_categories)

#获取电影名的特征向量

pool\_layer\_flat, dropout\_layer = get\_movie\_cnn\_layer(movie\_titles)

#得到电影特征

movie\_combine\_layer, movie\_combine\_layer\_flat = get\_movie\_feature\_layer(movie\_id\_embed\_layer, movie\_categories\_embed\_layer, dropout\_layer)

with tf.name\_scope("inference"):

#将用户特征和电影特征做矩阵乘法得到一个预测评分

inference = tf.reduce\_sum(user\_combine\_layer\_flat \* movie\_combine\_layer\_flat, axis=1)

inference = tf.expand\_dims(inference, axis=1)

with tf.name\_scope("loss"):

#MSE损失，将计算值回归到评分

cost = tf.losses.mean\_squared\_error(targets, inference )

loss = tf.reduce\_mean(cost)

#优化损失

global\_step = tf.Variable(0, name="global\_step", trainable=False)

optimizer = tf.train.AdamOptimizer(lr)

gradients = optimizer.compute\_gradients(loss) #代价

train\_op=optimizer.apply\_gradients(gradients,global\_step=global\_step)

4.模型训练

定义超参数：

#训练迭代次数

num\_epochs = 5

#每个Batch大小

batch\_size = 256

#丢弃率

dropout\_keep = 0.5

#学习率

learning\_rate = 0.0001

#每n 个batches 显示信息

show\_every\_n\_batches = 20

#保存路径

save\_dir = './save'

#定义取得batch的函数

def get\_batches(Xs, ys, batch\_size):

for start in range(0, len(Xs), batch\_size):

end = min(start + batch\_size, len(Xs))

yield Xs[start:end], ys[start:end]

#定义保存参数的函数

import pickle

def save\_params(params):

#保存参数到文件中

pickle.dump(params, open('params.p', 'wb'))

def load\_params():

#从文件中加载参数

return pickle.load(open('params.p', mode='rb'))

#作图

%matplotlib inline

%config InlineBackend.figure\_format = 'retina'

import matplotlib.pyplot as plt

import time

import datetime

#记录损失，用于画图

losses = {'train':[], 'test':[]}

with tf.Session(graph=train\_graph) as sess:

#搜集数据给TensorBoard使用

#跟踪渐变值和稀疏度

grad\_summaries = []

for g, v in gradients:

if g is not None:

grad\_hist\_summary = tf.summary.histogram("{}/grad/hist".format(v.name.replace(':', '\_')), g)

sparsity\_summary = tf.summary.scalar("{}/grad/sparsity".format(v.name.replace(':', '\_')), tf.nn.zero\_fraction(g))

grad\_summaries.append(grad\_hist\_summary)

grad\_summaries.append(sparsity\_summary)

grad\_summaries\_merged = tf.summary.merge(grad\_summaries)

#输出文件夹

timestamp = str(int(time.time()))

out\_dir = os.path.abspath(os.path.join(os.path.curdir, "runs", timestamp))

print("Writing to {}\n".format(out\_dir))

#损失与精度的总结

loss\_summary = tf.summary.scalar("loss", loss)

#训练的总结

train\_summary\_op = tf.summary.merge([loss\_summary, grad\_summaries\_merged])

train\_summary\_dir = os.path.join(out\_dir, "summaries", "train")

train\_summary\_writer = tf.summary.FileWriter(train\_summary\_dir, sess.graph)

#测试总结

inference\_summary\_op = tf.summary.merge([loss\_summary])

inference\_summary\_dir = os.path.join(out\_dir,"summaries","inference")

inference\_summary\_writer = tf.summary.FileWriter(inference\_summary\_dir, sess.graph)

#变量初始化

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

#模型保存

saver = tf.train.Saver()

for epoch\_i in range(num\_epochs):

#将数据集分成训练集和测试集，随机种子不固定

train\_X,test\_X, train\_y, test\_y = train\_test\_split(features,

targets\_values,

test\_size = 0.2,

random\_state = 0)

#分开batches

train\_batches = get\_batches(train\_X, train\_y, batch\_size)

test\_batches = get\_batches(test\_X, test\_y, batch\_size)

#训练的迭代，保存训练损失

for batch\_i in range(len(train\_X) // batch\_size):

x, y = next(train\_batches)

categories = np.zeros([batch\_size, 18])

for i in range(batch\_size):

categories[i] = x.take(6,1)[i]

titles = np.zeros([batch\_size, sentences\_size])

for i in range(batch\_size):

titles[i] = x.take(5,1)[i]

#传入数据

feed = {

uid: np.reshape(x.take(0,1), [batch\_size, 1]),

user\_gender: np.reshape(x.take(2,1), [batch\_size, 1]),

user\_age: np.reshape(x.take(3,1), [batch\_size, 1]),

user\_job: np.reshape(x.take(4,1), [batch\_size, 1]),

movie\_id: np.reshape(x.take(1,1), [batch\_size, 1]),

movie\_categories: categories, #x.take(6,1)

movie\_titles: titles, #x.take(5,1)

targets: np.reshape(y, [batch\_size, 1]),

dropout\_keep\_prob: dropout\_keep, #dropout\_keep

lr: learning\_rate}

#计算结果

step, train\_loss, summaries, \_ = sess.run([global\_step, loss, train\_summary\_op, train\_op], feed) #cost

losses['train'].append(train\_loss)

#保存记录

train\_summary\_writer.add\_summary(summaries, step)

#每多少个batches显示一次

if (epoch\_i \* (len(train\_X) // batch\_size) + batch\_i) % show\_every\_n\_batches == 0:

time\_str = datetime.datetime.now().isoformat()

print('{}: Epoch {:>3} Batch {:>4}/{} train\_loss = {:.3f}'.format(

time\_str,

epoch\_i,

batch\_i,

(len(train\_X) // batch\_size),

train\_loss))

#使用测试数据的迭代

for batch\_i in range(len(test\_X) // batch\_size):

x, y = next(test\_batches)

categories = np.zeros([batch\_size, 18])

for i in range(batch\_size):

categories[i] = x.take(6,1)[i]

titles = np.zeros([batch\_size, sentences\_size])

for i in range(batch\_size):

titles[i] = x.take(5,1)[i]

#传入数据

feed = {

uid: np.reshape(x.take(0,1), [batch\_size, 1]),

user\_gender: np.reshape(x.take(2,1), [batch\_size, 1]),

user\_age: np.reshape(x.take(3,1), [batch\_size, 1]),

user\_job: np.reshape(x.take(4,1), [batch\_size, 1]),

movie\_id: np.reshape(x.take(1,1), [batch\_size, 1]),

movie\_categories: categories, #x.take(6,1)

movie\_titles: titles, #x.take(5,1)

targets: np.reshape(y, [batch\_size, 1]),

dropout\_keep\_prob: 1,

lr: learning\_rate}

#计算结果

step, test\_loss, summaries = sess.run([global\_step, loss, inference\_summary\_op], feed) #cost

#保存测试损失

losses['test'].append(test\_loss)

inference\_summary\_writer.add\_summary(summaries, step)

#每多少个batches显示一次

time\_str = datetime.datetime.now().isoformat()

if (epoch\_i \* (len(test\_X) // batch\_size) + batch\_i) % show\_every\_n\_batches == 0:

print('{}: Epoch {:>3} Batch {:>4}/{} test\_loss = {:.3f}'.format(

time\_str,

epoch\_i,

batch\_i,

(len(test\_X) // batch\_size),

test\_loss))

#保存模型

saver.save(sess, save\_dir)

print('Model Trained and Saved')

相关代码如下：

#保存参数

save\_params((save\_dir))

load\_dir = load\_params()

#作图画出训练损失

plt.figure(figsize=(8, 6))

plt.plot(losses['train'], label='Training loss')

plt.legend()

plt.xlabel("Batches")

plt.ylabel("Loss")

\_ = plt.ylim()

#作图画出测试损失

plt.figure(figsize=(8,6))

plt.plot(losses['test'], label='Test loss')

plt.legend()

plt.xlabel("Batches")

plt.ylabel("Loss")

\_ = plt.ylim()

5.获取特征矩阵

1)定义函数用于获取保存的张量

相关代码如下：

def get\_tensors(loaded\_graph):

#使用get\_tensor\_by\_name()函数从loaded\_graph模块中获取张量

uid = loaded\_graph.get\_tensor\_by\_name("uid:0")

user\_gender = loaded\_graph.get\_tensor\_by\_name("user\_gender:0")

user\_age = loaded\_graph.get\_tensor\_by\_name("user\_age:0")

user\_job = loaded\_graph.get\_tensor\_by\_name("user\_job:0")

movie\_id = loaded\_graph.get\_tensor\_by\_name("movie\_id:0")

movie\_categories=loaded\_graph.get\_tensor\_by\_name("movie\_categories:0")

movie\_titles = loaded\_graph.get\_tensor\_by\_name("movie\_titles:0")

targets = loaded\_graph.get\_tensor\_by\_name("targets:0")

dropout\_keep\_prob = loaded\_graph.get\_tensor\_by\_name("dropout\_keep\_prob:0")

lr = loaded\_graph.get\_tensor\_by\_name("LearningRate:0")

inference = loaded\_graph.get\_tensor\_by\_name("inference/ExpandDims:0")

movie\_combine\_layer\_flat = loaded\_graph.get\_tensor\_by\_name("movie\_fc/Reshape:0")

user\_combine\_layer\_flat = loaded\_graph.get\_tensor\_by\_name("user\_fc/Reshape:0")

return uid, user\_gender, user\_age, user\_job, movie\_id, movie\_categories, movie\_titles, targets, lr, dropout\_keep\_prob, inference, movie\_combine\_layer\_flat, user\_combine\_layer\_flat

2)生成电影特征矩阵

相关代码如下：

loaded\_graph = tf.Graph()

movie\_matrics = []

with tf.Session(graph=loaded\_graph) as sess:

#载入保存好的模型

loader = tf.train.import\_meta\_graph(load\_dir + '.meta')

loader.restore(sess, load\_dir)

#调用函数提取tensors

uid, user\_gender, user\_age, user\_job, movie\_id, movie\_categories, movie\_titles, targets, lr, dropout\_keep\_prob, \_, movie\_combine\_layer\_flat, \_\_ = get\_tensors(loaded\_graph)

for item in movies.values:

categories = np.zeros([1, 18])

categories[0] = item.take(2)

titles = np.zeros([1, sentences\_size])

titles[0] = item.take(1)

feed = {

movie\_id: np.reshape(item.take(0), [1, 1]),

movie\_categories: categories,

movie\_titles: titles,

dropout\_keep\_prob: 1}

movie\_combine\_layer\_flat\_val = sess.run([movie\_combine\_layer\_flat], feed)

#添加进一个list中

movie\_matrics.append(movie\_combine\_layer\_flat\_val)

#保存成.p文件

pickle.dump((np.array(movie\_matrics).reshape(-1, 200)), open('movie\_matrics.p', 'wb'))

#读取文件

movie\_matrics = pickle.load(open('movie\_matrics.p', mode='rb'))

3)生成用户特征矩阵

相关代码如下：

loaded\_graph = tf.Graph()

users\_matrics = []

with tf.Session(graph=loaded\_graph) as sess:

#载入保存好的模型

loader = tf.train.import\_meta\_graph(load\_dir + '.meta')

loader.restore(sess, load\_dir)

#调用函数提取张量

uid, user\_gender, user\_age, user\_job, movie\_id, movie\_categories, movie\_titles, targets, lr, dropout\_keep\_prob, \_, \_\_,user\_combine\_layer\_flat = get\_tensors(loaded\_graph) #loaded\_graph

for item in users.values:

feed = {

uid: np.reshape(item.take(0), [1, 1]),

user\_gender: np.reshape(item.take(1), [1, 1]),

user\_age: np.reshape(item.take(2), [1, 1]),

user\_job: np.reshape(item.take(3), [1, 1]),

dropout\_keep\_prob: 1}

user\_combine\_layer\_flat\_val=sess.run([user\_combine\_layer\_flat], feed)

#添加进一个list中

users\_matrics.append(user\_combine\_layer\_flat\_val)

#保存成.p文件

pickle.dump((np.array(users\_matrics).reshape(-1, 200)), open('users\_matrics.p', 'wb'))

#读取文件

users\_matrics = pickle.load(open('users\_matrics.p', mode='rb'))

4.3.2 后端Django

1.路由文件

./mysite/urls.py的相关代码如下：

from django.contrib import admin

from django.urls import include, path

urlpatterns = [

#转发至connTest应用

path('connTest/', include('connTest.urls')),

#管理功能，默认生成，不使用

path('admin/', admin.site.urls),

]

如果请求中包含connTest/，则将请求转发至应用connTest的urls.py文件，./connTest/urls.py相关代码如下：

from django.urls import path

#导入connTest下的视图层views.py

from . import views

urlpatterns = [

#请求中不包含其他字符，调用视图层中的index函数

path('', views.index, name='index'),

#请求中包含“get\_rand\_movies/”，调用视图层中的get\_rand\_movies()函数

path('get\_rand\_movies/',views.get\_rand\_movies,name='get\_rand\_movies'),

#请求中包含“get\_this\_movie/”，调用视图层中的get\_this\_movie()函数

path('get\_this\_movie/', views.get\_this\_movie, name='get\_this\_movie'),

#请求中包含“post\_st\_movies/”，调用视图层中的post\_st\_movies()函数

path('post\_st\_movies/', views.post\_st\_movies, name='post\_st\_movies'),

#请求中包含“post\_of\_movies/”，调用视图层中的post\_of\_movies()函数

path('post\_of\_movies/',views.post\_of\_movies,name='post\_of\_movies'),

]

如果请求中有对应的字符串，则转发至相应的处理函数。

2.视图层文件

相关代码如下：

#用于返回应答

from django.http import HttpResponse

#自定义文件my\_data.py，用于封装推荐算法函数

from . import my\_data

#路径

import os

#特征向量文件所在目录所需文件settings.PROJECT

from django.conf import settings

#处理json文件，API返回格式为json

import json

#index函数，接收视图层请求，此函数用于测试，推荐算法中不使用

def index(request):

#如果请求方法是POST

if(request.method == 'POST'):

data = request.POST['choice']

#返回get\_random\_movies()结果

rand\_movies\_list=my\_data.get\_random\_movies(settings.PROJECT\_ROOT)

results = {}

for i in range(5):

result = {}

result['movie\_id'] = rand\_movies\_list[i][0]

result['movie\_name'] = rand\_movies\_list[i][1]

result['movie\_genres'] = rand\_movies\_list[i][2]

results[str(i)] = result

results = json.dumps(results)

return HttpResponse(results)

elif(request.method == 'GET'):

#如果请求方法是GET

return HttpResponse("Hello,this is connTest index.")

#get\_rand\_movies函数，接收视图层请求，返回5个随机电影

def get\_rand\_movies(request):

#调用my\_data中的get\_rand\_movies()函数，获得电影

rand\_movies\_list = my\_data.get\_random\_movies(settings.PROJECT\_ROOT)

results = {}

#遍历list化成dict

for i in range(5):

result = {}

result['movie\_id'] = rand\_movies\_list[i][0]

result['movie\_name'] = rand\_movies\_list[i][1]

result['movie\_genres'] = rand\_movies\_list[i][2]

results[str(i)] = result

#转化为json格式

results = json.dumps(results)

#返回应答

return HttpResponse(results)

#get\_this\_movie函数，接收视图层请求，返回电影相关信息，系统中未使用，仅做测试使用

def get\_this\_movie(request):

#POST请求，接收参数movie\_id

movie\_id = request.POST['movie\_id']

movie\_id = int(movie\_id)

#调用my\_data中的get\_a\_movie()函数，获得电影

this\_movie\_data = my\_data.get\_a\_movie(settings.PROJECT\_ROOT,movie\_id)

#转化成dict

results = {}

result = {}

result['movie\_id'] = this\_movie\_data[0]

result['movie\_name'] = this\_movie\_data[1]

result['movie\_genres'] = this\_movie\_data[2]

results["0"] = result

#转化成json格式

results = json.dumps(results)

#返回应答

return HttpResponse(results)

#post\_st\_movies()函数，接收视图层请求，返回某个电影的同种电影

def post\_st\_movies(request):

#接收POST请求参数

movie\_id = request.POST['movie\_id']

movie\_id = int(movie\_id)

#调用my\_data中的recommend\_same\_type\_movie()函数，获得同种电影列表

st\_movies\_list = my\_data.recommend\_same\_type\_movie(settings.PROJECT\_ROOT,movie\_id)

#list化成dict

results = {}

for i in range(5):

result = {}

result['movie\_id'] = st\_movies\_list[i][0]

result['movie\_name'] = st\_movies\_list[i][1]

result['movie\_genres'] = st\_movies\_list[i][2]

results[str(i)] = result

#转化成json格式

results = json.dumps(results)

#返回应答

return HttpResponse(results)

#post\_of\_movies函数，接收视图层请求，返回看过某个电影的人喜欢的电影

def post\_of\_movies(request):

#接收POST请求参数

movie\_id = request.POST['movie\_id']

movie\_id = int(movie\_id)

#调用my\_data中的recommend\_other\_favorite\_movie ()函数，获得电影列表

of\_movies\_list = my\_data.recommend\_other\_favorite\_movie(settings.PROJECT\_ROOT,movie\_id)

#list转化成dict

results = {}

for i in range(5):

result = {}

result['movie\_id'] = of\_movies\_list[i][0]

result['movie\_name'] = of\_movies\_list[i][1]

result['movie\_genres'] = of\_movies\_list[i][2]

results[str(i)] = result

#转化成json格式

results = json.dumps(results)

#返回应答

return HttpResponse(results)

实现推荐算法文件my\_data.py相关代码如下：

#导入需要用到的包

import pandas as pd

import numpy as np

import pickle

import random

import os

#get\_random\_movies函数返回5个随机电影

def get\_random\_movies(PROJECT\_ROOT):

#读取数据

title\_count, title\_set, genres2int, features, targets\_values, ratings, users, movies, data, movies\_orig, users\_orig = pickle.load(open(os.path.join(PROJECT\_ROOT,'preprocess.p'), mode='rb'))

#随机5个索引

random\_movies = [random.randint(0,3833) for i in range(5)]

#返回电影数据列表

return movies\_orig[random\_movies]

#get\_a\_movie函数，参数电影ID对应信息

def get\_a\_movie(PROJECT\_ROOT,movie\_id):

title\_count, title\_set, genres2int, features, targets\_values, ratings, users, movies, data, movies\_orig, users\_orig = pickle.load(open(os.path.join(PROJECT\_ROOT,'preprocess.p'), mode='rb'))

#电影ID转下标的字典，数据集中电影ID跟下标不一致，例如，第5行的数据电影ID不一定是5

movieid2idx = {val[0]:i for i, val in enumerate(movies.values)}

this\_movie\_data = movies\_orig[movieid2idx[movie\_id]]

return this\_movie\_data

#recommend\_same\_type\_movie函数返回同种类电影列表

def recommend\_same\_type\_movie(PROJECT\_ROOT,movie\_id\_val, top\_k=20):

#读取数据

title\_count, title\_set, genres2int, features, targets\_values, ratings, users, movies, data, movies\_orig, users\_orig = pickle.load(open(os.path.join(PROJECT\_ROOT,'preprocess.p'), mode='rb'))

#电影ID转下标的字典，数据集中电影ID跟下标不一致，例如，第5行的数据电影ID不一定是5

movieid2idx = {val[0]:i for i, val in enumerate(movies.values)}

#读取电影特征矩阵 movie\_matrics=pickle.load(open(os.path.join(PROJECT\_ROOT,'movie\_matrics.p'), mode='rb'))

#读取用户特征矩阵

users\_matrics = pickle.load(open(os.path.join(PROJECT\_ROOT,'users\_matrics.p'), mode='rb'))

#推荐与选择

同类型的电影

print("您选择的电影是：{}".format(movies\_orig[movieid2idx[movie\_id\_val]]))

#规范化电影特征矩阵

norm\_movie\_matrics = np.sqrt(np.sum(np.square(movie\_matrics),axis=1)).reshape(3883,1)

normalized\_movie\_matrics = movie\_matrics/norm\_movie\_matrics

#获取所选电影特征向量

probs\_embeddings = (movie\_matrics[movieid2idx[movie\_id\_val]]).reshape([1, 200])

probs\_embeddings = probs\_embeddings/np.sqrt(np.sum(np.square(probs\_embeddings)))

#计算相似度

probs\_similarity = np.matmul(probs\_embeddings, np.transpose(normalized\_movie\_matrics))

#print("根据您看的电影类型给您的推荐：")

p = np.squeeze(probs\_similarity)

#获取topk个电影

p[np.argsort(p)[:-top\_k]] = 0

p = p / np.sum(p)

results = set()

#在topk个电影汇总选取5个

while len(results) != 5:

c = np.random.choice(3883, 1, p=p)[0]

results.add(c)

final\_results = [movies\_orig[val] for val in results]

#返回电影列表

return final\_results

#recommend\_other\_favorite\_movie函数，返回看过同一个电影的人喜欢的电影

def recommend\_other\_favorite\_movie(PROJECT\_ROOT,movie\_id\_val, top\_k=20):

#读取数据

title\_count, title\_set, genres2int, features, targets\_values, ratings, users, movies, data, movies\_orig, users\_orig = pickle.load(open(os.path.join(PROJECT\_ROOT,'preprocess.p'), mode='rb'))

#电影ID转下标的字典，数据集中电影ID跟下标不一致，例如，第5行的数据电影ID不一定是5

movieid2idx = {val[0]:i for i, val in enumerate(movies.values)}

#读取电影特征与用户特征矩阵

movie\_matrics = pickle.load(open(os.path.join(PROJECT\_ROOT,'movie\_matrics.p'), mode='rb'))

users\_matrics = pickle.load(open(os.path.join(PROJECT\_ROOT,'users\_matrics.p'), mode='rb'))

#推荐看过同一个的电影的人喜欢的电影

print("您看的电影是：{}".format(movies\_orig[movieid2idx[movie\_id\_val]]))

#根据电影寻找相似的人

probs\_movie\_embeddings = (movie\_matrics[movieid2idx[movie\_id\_val]]).reshape([1, 200])

probs\_movie\_embeddings = probs\_movie\_embeddings/np.sqrt(np.sum(np.square(probs\_movie\_embeddings)))

norm\_users\_matrics = np.sqrt(np.sum(np.square(users\_matrics),axis=1)).reshape(6040,1)

normalized\_users\_matrics = users\_matrics/norm\_users\_matrics

#计算相似度

probs\_user\_favorite\_similarity = np.matmul(probs\_movie\_embeddings, np.transpose(normalized\_users\_matrics))

favorite\_user\_id = np.argsort(probs\_user\_favorite\_similarity)[0][-top\_k:]

print("喜欢看这个电影的人是：{}".format(users\_orig[favorite\_user\_id-1]))

#他们喜欢什么样的电影

probs\_users\_embeddings = (users\_matrics[favorite\_user\_id-1]).reshape([-1, 200])

probs\_users\_embeddings = probs\_users\_embeddings/np.sqrt(np.sum(np.square(probs\_users\_embeddings)))

norm\_movie\_matrics = np.sqrt(np.sum(np.square(movie\_matrics),axis=1)).reshape(3883,1)

normalized\_movie\_matrics = movie\_matrics/norm\_movie\_matrics

#计算相似度

probs\_similarity = np.matmul(probs\_users\_embeddings, np.transpose(normalized\_movie\_matrics))

p = np.argmax(probs\_similarity, 1)

#print("喜欢看这个电影的人还喜欢看：")

results = set()

#随机选取5个

while len(results) != 5:

c = p[random.randrange(top\_k)]

results.add(c)

final\_results = [movies\_orig[val] for val in results]

return final\_results

3.项目设置文件

相关代码如下：

#用于路径

import os

#默认设置

BASE\_DIR = os.path.dirname(os.path.dirname(os.path.abspath(\_\_file\_\_)))

SECRET\_KEY = '' #项目的秘钥，这里不显示

DEBUG = False

#允许接入的用户，通常填写域名，例如：

#ALLOWED\_HOSTS = ['www.baidu.com']

ALLOWED\_HOSTS = ['’] # 这里不显示

#Application definition

INSTALLED\_APPS = [

#添加应用connTest

'connTest',

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

]

MIDDLEWARE = [

'django.middleware.security.SecurityMiddleware',

'django.contrib.sessions.middleware.SessionMiddleware',

'django.middleware.common.CommonMiddleware',

#如果报错，直接注释掉即可

#'django.middleware.csrf.CsrfViewMiddleware',

'django.contrib.auth.middleware.AuthenticationMiddleware',

'django.contrib.messages.middleware.MessageMiddleware',

'django.middleware.clickjacking.XFrameOptionsMiddleware',

]

ROOT\_URLCONF = 'mysite.urls'

PROJECT\_ROOT = os.path.abspath(os.path.dirname(\_\_file\_\_))

TEMPLATES = [

{

'BACKEND': 'django.template.backends.django.DjangoTemplates',

'DIRS': [os.path.join(BASE\_DIR, 'templates')],

'APP\_DIRS': True,

'OPTIONS': {

'context\_processors': [

'django.template.context\_processors.debug',

'django.template.context\_processors.request',

'django.contrib.auth.context\_processors.auth',

'django.contrib.messages.context\_processors.messages',

],

},

},

]

WSGI\_APPLICATION = 'mysite.wsgi.application'

#数据库

#参考https://docs.djangoproject.com/en/2.2/ref/settings/#databases

#数据库在这更换，此处未用到数据库，所以不需更换

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.sqlite3',

'NAME': os.path.join(BASE\_DIR, 'db.sqlite3'),

}

}

#密码验证

AUTH\_PASSWORD\_VALIDATORS = [

{

'NAME': 'django.contrib.auth.password\_validation.UserAttributeSimilarityValidator',

},

{

'NAME': 'django.contrib.auth.password\_validation.MinimumLengthValidator',

},

{

'NAME': 'django.contrib.auth.password\_validation.CommonPasswordValidator',

},

{

'NAME': 'django.contrib.auth.password\_validation.NumericPasswordValidator',

},

]

#语言

LANGUAGE\_CODE = 'en-us'

#时区

TIME\_ZONE = 'UTC'

USE\_I18N = True

USE\_L10N = True

USE\_TZ = True

STATIC\_URL = '/static/'

4.3.3 前端微信小程序

1.小程序全局配置文件

app.js相关代码如下：

//app.js

App({

onLaunch: function () {

console.log("app launch")

//展示本地存储能力

var logs = wx.getStorageSync('logs') || []

logs.unshift(Date.now())

wx.setStorageSync('logs', logs)

//登录

wx.login({

success: res => {

}

});

//获取用户信息

wx.getSetting({

success: res => {

if (res.authSetting['scope.userInfo']) {

//已经授权，可以直接调用 getUserInfo 获取头像昵称，不会弹框

wx.getUserInfo({

success: res => {

//可以将res发送给后台解码出unionId

this.globalData.userInfo = res.userInfo

//由于 getUserInfo 是网络请求，可能会在 Page.onLoad 之后才返回

//此处加入callback 以防止类似情况

if (this.userInfoReadyCallback) {

this.userInfoReadyCallback(res)

}

}

})

}

}

})

},

globalData: {

userInfo: null,

}

})

文件app.json为json格式，不能添加注释。相关代码如下：

{

"pages": [

"pages/movies/movies",

"pages/index/index",

"pages/logs/logs"

],

"window": {

"backgroundTextStyle": "light",

"navigationBarBackgroundColor": "#fff",

"navigationBarTitleText": "Movies",

"navigationBarTextStyle": "black"

},

"tabBar": {

"list": [

{

"pagePath": "pages/movies/movies",

"text": "Movies",

"iconPath": "/icon/movie.png",

"selectedIconPath": "/icon/movie\_selected.png"

},

{

"pagePath": "pages/index/index",

"text": "Mine",

"iconPath": "/icon/user.png",

"selectedIconPath": "/icon/user\_selected.png"

}

]

},

"style": "v2",

"sitemapLocation": "sitemap.json"

}

文件app.wxss中描述了小程序的样式表，用于配置全局页面元素样式，app.wxss相关代码如下：

/\*\*app.wxss\*\*/

@import './weui-miniprogram/weui-wxss/dist/style/weui.wxss';

/\*定义了container的样式\*/

.container {

height: 100%;

display: flex;

flex-direction: column;

align-items: center;

justify-content: space-between;

padding: 200rpx 0;

box-sizing: border-box;

}

2.推荐电影页面

相关代码如下：

//pages/movies/movies.js

Page({

/\*页面的初始数据\*/

data: {

//电影相关信息

movies: null,

//推荐方式

recommend\_mode: null,

//推荐方式选择信息

recommend\_mode\_list: [{

name: "同类型电影", value: 0, checked: "false"

},

{

name: "看过这个的还喜欢看", value: 1, checked: "true"

}]

},

//页面单选按钮逻辑功能函数，选择推荐方式时触发

radioChange: function (e) {

//用单选按钮的值给推荐方式变量赋值

this.setData({ recommend\_mode: e.detail.value })

console.log(this.data.recommend\_mode)

},

//推荐同类型电影

post\_st\_movies: function (event) {

var that = this;

//向后端服务器发出请求

wx.request({

//这里url填写用户服务器的域名，不显示

url: '',

//发送数据为电影的ID

data: {

movie\_id: event.currentTarget.dataset.movie\_id,

},

//POST方法

method: 'post',

header: {

'content-type': 'application/x-www-form-urlencoded' //默认值

},

//成功后执行

success(res) {

//更新电影信息

that.setData({ movies: res.data })

console.log(that.data.movies[0].movie\_id);

}

})

},

//推荐看过这个电影的人喜欢的电影

post\_of\_movies: function (event) {

var that = this;

//向后端服务器发出请求

wx.request({

//这里url填写用户服务器的域名，不显示

url: '',

//发送数据为电影的ID

data: {

movie\_id: event.currentTarget.dataset.movie\_id,

},

//POST方法

method: 'post',

header: {

'content-type': 'application/x-www-form-urlencoded' //默认值

},

//成功后执行

success(res) {

//更新电影信息

that.setData({ movies: res.data })

console.log(that.data.movies[0].movie\_id);

}

})

},

//电影推荐按钮的逻辑功能

recommend\_movies: function (event) {

//如果是0模式

if (this.data.recommend\_mode == 0) {

this.post\_st\_movies(event);

}

//如果是1模式

else if (this.data.recommend\_mode == 1) {

this.post\_of\_movies(event);

}

},

//随机获取电影函数逻辑功能

get\_rand\_movies: function () {

var that = this;

//向后端服务器发出请求

wx.request({

//url填写用户服务器的域名，不显示

url: '',

data: {

},

//GET方法

method: 'get',

header: {

'content-type': 'application/x-www-form-urlencoded' //默认值

},

//成功后执行

success(res) {

//更新电影信息

that.setData({ movies: res.data })

console.log(that.data.movies[0].movie\_id);

}

})

},

/\*生命周期函数--监听页面加载\*/

onLoad: function (options) {

this.get\_rand\_movies();

},

/\*生命周期函数--监听页面初次渲染完成\*/

onReady: function () {

},

/\* 生命周期函数--监听页面显示\*/

onShow: function () {

},

/\*生命周期函数--监听页面隐藏\*/

onHide: function () {

},

/\*生命周期函数--监听页面卸载\*/

onUnload: function () {

},

/\*页面相关事件处理函数--监听用户下拉动作\*/

onPullDownRefresh: function () {

},

/\*页面上拉触底事件的处理函数\*/

onReachBottom: function () {

},

/\*用户单击右上角分享\*/

onShareAppMessage: function () {

}

})

//文件movies.json代码为空，movies.wxml相关代码如下：

<!--pages/movies/movies.wxml-->

<!--页面容器-->

<view class="container">

<!--页面-->

<view class="page-body">

<!--推荐方式选择容器-->

<view class="mode-choose-container">

<text>请选择推荐方式：</text>

<radio-group class="mode-choose" bindchange="radioChange">

<radio class="radio" wx:for-items="{{recommend\_mode\_list}}" value="{{item.value}}" >

<text>{{item.name}}</text>

</radio>

</radio-group>

</view>

<!--电影信息容器-->

<view class="movie-container" wx:for="{{[0,1,2,3,4]}}" hover-class='hover\_list' data-movie\_id="{{movies[index].movie\_id}}" bindtap="recommend\_movies">

<text>Top {{index+1}}</text>

<!--电影名称-->

<view class="movie-name-container">

<text class="movie-name">

{{movies[index].movie\_name}}

</text>

</view>

<!--电影流派-->

<view class="movie-genres-container">

<text class="movie-genres">

{{movies[index].movie\_genres}}

</text>

</view>

</view>

</view>

</view>

//movies.wxss相关代码

/\* pages/movies/movies.wxss \*/

.container {

/\*透明度: 0.1\*/

align-items: center;

background: #f5f5f5;

}

.page-body {

/\*透明度:0.2\*/

align-items: center;

background: #fefefe;

border-style: solid;

border-color: #b2b2b2;

border-width: thin medium medium thin;

border-radius: 50rpx;

width: 660rpx;

}

.mode-choose-container {

align-items: center;

padding:20rpx;

}

.movie-container {

align-items: center;

border-style: solid;

border-color: #b2b2b2;

border-width: thin medium medium thin;

border-radius: 50rpx;

margin: 20rpx 15rpx 20rpx 15rpx;

padding:20rpx;

text-align: center;

}

.hover\_list {

opacity: 0.9;

background: #f7f7f7;

}

3.个人信息界面以及用户登录记录页面

index.js相关代码如下：

//获取应用实例

const app = getApp()

Page({

data: {

motto: 'Hope you find your peace.',

userInfo: {},

hasUserInfo: false,

canIUse: wx.canIUse('button.open-type.getUserInfo')

},

//事件处理函数

bindViewTap: function() {

wx.navigateTo({

url: '../logs/logs'

})

},

onLoad: function () {

if (app.globalData.userInfo) {

this.setData({

userInfo: app.globalData.userInfo,

hasUserInfo: true

})

} else if (this.data.canIUse){

//由于getUserInfo是网络请求，可能会在Page.onLoad后才返回

//此处加入callback 防止这种情况发生

app.userInfoReadyCallback = res => {

this.setData({

userInfo: res.userInfo,

hasUserInfo: true

})

}

} else {

//在没有 open-type=getUserInfo版本的兼容处理

wx.getUserInfo({

success: res => {

app.globalData.userInfo = res.userInfo

this.setData({

userInfo: res.userInfo,

hasUserInfo: true

})

}

})

}

},

getUserInfo: function(e) {

console.log(e)

app.globalData.userInfo = e.detail.userInfo

this.setData({

userInfo: e.detail.userInfo,

hasUserInfo: true

})

}

})

//index.html相关代码

<!--index.wxml-->

<view class="container">

<view class="userinfo">

<button wx:if="{{!hasUserInfo && canIUse}}" open-type="getUserInfo" bindgetuserinfo="getUserInfo"> 获取头像昵称 </button>

<block wx:else>

<image bindtap="bindViewTap" class="userinfo-avatar" src="{{userInfo.avatarUrl}}" mode="cover"></image>

<text class="userinfo-nickname">{{userInfo.nickName}}</text>

</block>

</view>

<view class="usermotto">

<text class="user-motto">{{motto}}</text>

</view>

</view>

//文件index.json为空，index.wxss相关代码

/\*\*index.wxss\*\*/

.userinfo {

display: flex;

flex-direction: column;

align-items: center;

}

.userinfo-avatar {

width: 128rpx;

height: 128rpx;

margin: 20rpx;

border-radius: 50%;

}

.userinfo-nickname {

color: #aaa;

}

.usermotto {

margin-top: 200px;

}

logs.js相关代码如下：

//logs.js

const util = require('../../utils/util.js')

Page({

data: {

logs: []

},

onLoad: function () {

this.setData({

logs: (wx.getStorageSync('logs') || []).map(log => {

return util.formatTime(new Date(log))

})

})

}

})

//logs.html相关代码

<!--logs.wxml-->

<view class="container log-list">

<block wx:for="{{logs}}" wx:for-item="log">

<text class="log-item">{{index + 1}}. {{log}}</text>

</block>

</view>

//logs.wxss代码如下：

.log-list {

display: flex;

flex-direction: column;

padding: 40rpx;

}

.log-item {

margin: 10rpx;

}

//文件logs.json代码

{

"navigationBarTitleText": "查看启动日志",

"usingComponents": {}

}

项目5 基于排队时间预测的智能导航推荐系统

5.3 模块实现

5.3.1 数据预处理

1.加载数据集

相关代码如下：

import pandas as pd

#乘车刷卡交易数据表

train\_data = pd.read\_csv('C:/Users/99509/Desktop/信息系统设计/data/gd\_train\_data.txt',header=None,names=['use\_city','line\_name','terminal\_id','card\_id','create\_city','deal\_time','card\_type'])

#公交线路信息表

line\_desc = pd.read\_csv('C:/Users/99509/Desktop/信息系统设计/data/gd\_line\_desc.txt',header=None,names=['line\_name','stop\_cnt','line\_type'])

#广州市天气状况信息表

weather\_report = pd.read\_csv('C:/Users/99509/Desktop/信息系统设计/data/gd\_weather\_report.txt',header=None,names=['data','weather','temperature','wind\_direction\_force'])

#虽然公交线路数据表中有21条线路，但是数据集只包含2条线路的详细数据

print(train\_data)

2.时间划分与保存

相关代码如下：

for i in ['线路6', '线路11']:

train\_data\_lineX = train\_data[train\_data['line\_name'] == i]

#取原始乘车表中对应线路符合的数据

#把交易数据的日期和小时分成两个字段

train\_data\_lineX['date'] = train\_data\_lineX['deal\_time'].apply(lambda x: str(x).split(' ')[0]) #日期

train\_data\_lineX['time'] = train\_data\_lineX['deal\_time'].apply(lambda x: int(str(x).split(' ')[1].split(':')[0])) #小时

train\_data\_lineX\_date\_time = train\_data\_lineX.drop('deal\_time', axis=1, inplace=False) #删除原始交易时间元素

#取交易时间在6~21点之间的数据

train\_data\_lineX\_date\_time\_06 = train\_data\_lineX\_date\_time[6 <= train\_data\_lineX\_date\_time['time']]

train\_data\_lineX\_date\_time\_06\_21 = train\_data\_lineX\_date\_time\_06[train\_data\_lineX\_date\_time\_06['time'] <= 21]

#数据按小时划分并累计单位小时内客流量

lineX\_passenger\_hour = DataFrame(train\_data\_lineX\_date\_time\_06\_21.groupby(['date', 'time']).count()['card\_id']).reset\_index()

#保存最终所需数据

if i == '线路6':

lineX\_passenger\_hour.to\_csv('C:/Users/99509/Desktop/信息系统设计/data/line6\_passenger\_hour.csv', header=1, index=0, encoding='utf-8')

if i == '线路11':

lineX\_passenger\_hour.to\_csv('C:/Users/99509/Desktop/信息系统设计/data/line11\_passenger\_hour.csv', header=1, index=0, encoding='utf-8')

3.处理天气预报数据

相关代码如下：

#日期标准化 eg：'10/10'转换为'10'，'10' '1/1'转换为'01','01'

def changeDate(date):

dateList = date.split('/')

if int(dateList[1]) < 10:

month = '0' + dateList[1]

else:

month = dateList[1]

if int(dateList[2]) < 10:

day = '0' + dateList[2]

else:

day = dateList[2]

return dateList[0] + month + day

#转换天气预报数组中的日期格式

weather\_report['datestr'] = weather\_report['date'].apply(lambda x: changeDate(x))

#将相应字段分开

weather\_report['weather\_d'] = weather\_report['weather'].apply(lambda x: x.split('/')[0]) #白天

weather\_report['weather\_n'] = weather\_report['weather'].apply(lambda x: x.split('/')[1]) #晚上

weather\_report['temperature\_h'] = weather\_report['temperature'].apply(lambda x: int(re.sub(r'\D', '', x.split('/')[0]))) #最高温

weather\_report['temperature\_l'] = weather\_report['temperature'].apply(lambda x: int(re.sub(r'\D', '', x.split('/')[1]))) #最低温

weather\_report['wind\_direction\_force\_d'] = weather\_report['wind\_direction\_force'].apply(lambda x: x.split('/')[0])

#白天风向、风力

weather\_report['wind\_direction\_force\_n'] = weather\_report['wind\_direction\_force'].apply(lambda x: x.split('/')[1])

#晚上风向、风力

weather\_report['temperature\_average'] = (weather\_report['temperature\_h'] + weather\_report['temperature\_l']) / 2.0 #平均温度

weather\_report['temperature\_abs'] = abs(weather\_report['temperature\_h'] - weather\_report['temperature\_l']) #温差

#降低风向影响，着重考虑风力影响，按照以下将风力、风向数字化

print(pd.concat([weather\_report['wind\_direction\_force\_d'], weather\_report['wind\_direction\_force\_n']],ignore\_index=True).drop\_duplicates()) #数组合并去冗余、得到所有的风力、风向种类

windmap = {'无持续风向≤3级': 0, '无持续风向微风转3-4级': 1, '北风微风转3-4级': 1, '东北风3-4级': 2, '北风3-4级': 2, '东南风3-4级': 2, '东风4-5级': 3,'北风4-5级': 3} #划分

weather\_report['wind\_direction\_force\_d\_map'] = weather\_report['wind\_direction\_force\_d'].map(windmap) #将原始数据替换为标准化数据

weather\_report['wind\_direction\_force\_n\_map'] = weather\_report['wind\_direction\_force\_n'].map(windmap) #将原始数据替换为标准化数据

weather\_report['wind\_average'] = (weather\_report['wind\_direction\_force\_d\_map'] + weather\_report['wind\_direction\_force\_n\_map']) / 2.0 #做风力、风向平均值

weather\_report['wind\_abs'] = abs(weather\_report['wind\_direction\_force\_d\_map'] - weather\_report['wind\_direction\_force\_n\_map']) #做风力、风向差

#将天气按照以下形式数字化

print(pd.concat([weather\_report['weather\_d'], weather\_report['weather\_n']], ignore\_index=True).drop\_duplicates())

#数据组合并去冗余，得到所有天气种类

weathermap = {'晴': 0, '多云': 1, '阴': 2, '小雨': 3, '小到中雨': 4, '中雨': 5, '中到大雨': 6, '大雨': 7, '大到暴雨': 8, '霾': 9, '阵雨': 10,'雷阵雨': 11}

#划分

weather\_report['weather\_d\_map'] = weather\_report['weather\_d'].map(weathermap) #将原始数据替换为标准化数据

weather\_report['weather\_n\_map'] = weather\_report['weather\_n'].map(weathermap) #将原始数据替换为标准化数据

weather\_report['weather\_average'] = (weather\_report['weather\_d\_map'] + weather\_report['weather\_n\_map']) / 2.0 #做天气平均值

weather\_report['weather\_abs'] = abs(weather\_report['weather\_d\_map'] - weather\_report['weather\_n\_map']) #做天气差

#去除多余项，留下标准化日期、风力风向、风力风向均值、风力风向差、天气、天气均值、天气差

weather\_report\_result = weather\_report.drop(['date', 'weather', 'temperature', 'wind\_direction\_force', 'weather\_d', 'weather\_n', 'wind\_direction\_force\_d','wind\_direction\_force\_n'], axis=1, inplace=False)

#输出

weather\_report\_result = weather\_report\_result.reset\_index(drop=True)

#重置索引

print(weather\_report\_result)

for i in range(len(weather\_report\_result)):

weather\_report\_result.loc[i, 'datestr'] = pd.to\_datetime(weather\_report\_result.loc[i, 'datestr'],format='%Y%m%d').strftime('%Y-%m-%d') #将时间格式化为year-month-day

weather\_report\_result.to\_csv('C:/Users/99509/Desktop/信息系统设计/data/weather\_report\_result.csv', header=0, index=0, encoding='utf-8')

4.增加特征

相关代码如下：

weather\_report\_data = pd.read\_csv('C:/Users/99509/Desktop/信息系统设计/data/weather\_report\_result.csv', header=None,names=['date', 'temperature\_h', 'temperature\_l', 'temperature\_average', 'temperature\_abs', 'wind\_d\_map', 'wind\_n\_map', 'wind\_average', 'wind\_abs', 'weather\_d\_map', 'weather\_n\_map', 'weather\_average', 'weather\_abs'])

holiday = pd.read\_csv('C:/Users/99509/Desktop/信息系统设计/data/date\_holiday.txt', header=None, names=['date', 'isholiday'])

#日期，是否是假期

testdate = pd.date\_range('2014-12-25', '2014-12-31') #测试集日期范围：2014.12.25-2014.12.31

#获得测试机完整日期时间范围

datetimelist = []

for idate in testdate:

datetime = pd.date\_range(str(idate) + ' 6:00', str(idate) + ' 21:00', freq='H') #测试集时间范围：6：00~21：00

datetimelist.append(DataFrame(datetime))

datetimeDf = pd.concat(datetimelist, ignore\_index=True) #忽略索引连接数组

datetimeDf.columns = ['datetime'] #修改列名为datetime

datetimeDf['date'] = datetimeDf['datetime'].apply(lambda x: str(x).split(' ')[0]) #分离出日期

datetimeDf['time'] = datetimeDf['datetime'].apply(lambda x: int(str(x).split(' ')[1].split(':')[0])) #分离出时间

datetimeDf\_date\_time = datetimeDf.drop('datetime', axis=1, inplace=False) #保存测试所有时间段：7天\*15小时

#判断是否为节假日，数据预处理

for i in ['6', '11']:

lineX\_passenger\_hour\_path = "C:/Users/99509/Desktop/信息系统设计/data/line%s\_passenger\_hour.csv" % i

#得到线路6/11的训练集中每小时客流量的路径（因为%s在字符串中，所以%i表示用i替换s）

lineX\_passenger\_hour = pd.read\_csv(lineX\_passenger\_hour\_path)

#得到线路6或11训练集中每小时的客流量

train\_passenger\_test = pd.concat([lineX\_passenger\_hour, datetimeDf\_date\_time], ignore\_index=True)

#连接测试集和训练集中线路6或11的每小时客流量

#取线路6或11的交集（得到完整数据：日期、最高温、最低温、平均温度、温差、标准化白天风力风向、标准化夜晚风力风向、平均风力风向、风力风向差、标准化白天天气、标准化夜晚天气、平均天气、天气差、客流量）

test\_data\_weather = pd.merge(train\_passenger\_test, weather\_report\_data, on='date', how='left')

test\_data\_weather\_holiday = pd.merge(test\_data\_weather, holiday, on='date', how='left') #测试集中节假日的数据

test\_data\_weather\_holiday['dayofweek'] = test\_data\_weather\_holiday['date'].apply(lambda x: pd.to\_datetime(x).dayofweek) #添加是否为周末元素至测试集

5.合并特征值

相关代码如下：

weather\_report\_data = pd.read\_csv('C:/Users/99509/Desktop/信息系统设计/data/weather\_report\_result.csv', header=None,names=['date', 'temperature\_h','temperature\_l','temperature\_average','temperature\_abs', 'wind\_d\_map','wind\_n\_map','wind\_average','wind\_abs','weather\_d\_map','weather\_n\_map','weather\_average','weather\_abs']) #读取数据8.1~1.31

line6\_weather = weather\_report\_data[:146] #读取数据8.1~12.24 (天)

line6\_dateinfo = pd.read\_csv('C:/Users/99509/Desktop/信息系统设计/data/line6\_train\_data\_no\_dum\_scale.csv',header=None,names=['card\_id','date','time','temperature\_h','temperature\_l','temperature\_average','temperature\_abs', 'wind\_d\_map','wind\_n\_map','wind\_average','wind\_abs','weather\_d\_map','weather\_n\_map','weather\_average','weather\_abs', 'isholiday','dayofweek' ])

#读取数据8.1~12.24 (小时)

#print(line6\_dateinfo)调试代码

line6\_weather\_date = pd.DataFrame()

line6\_weather\_date['card\_id']=line6\_dateinfo['card\_id'].drop(0)

line6\_weather\_date['time']=line6\_dateinfo['time']

line6\_weather\_date['date']=line6\_dateinfo['date']

line6\_weather\_date['isholiday']=line6\_dateinfo['isholiday']

line6\_weather\_date['dayofweek']=line6\_dateinfo['dayofweek']

line6\_weather\_date\_f=pd.merge(line6\_weather,line6\_weather\_date)

line6\_weather\_date\_final=line6\_weather\_date\_f.drop(['date', 'temperature\_h','temperature\_l','temperature\_abs','wind\_d\_map','wind\_n\_map','wind\_abs','weather\_d\_map','weather\_n\_map','weather\_abs'], axis=1,inplace=False)

line6\_weather\_date\_final.to\_csv('C:/Users/99509/Desktop/信息系统设计/修改/line6\_weather\_date\_final.csv',header=1, index=0, encoding='utf-8')

5.3.2 客流预测

1.创建并保存模型

使用程序gbdtmodel.py建立GBDT模型。

#建立GBDT模型

gbdt=ensemble.GradientBoostingRegressor(learning\_rate=0.5, n\_estimators=80, max\_depth=3) #用GridSearchCV进行调参后确定的参数

#print(cross\_val\_score(gbdt, features, lables, scoring=score, cv=cv))

gbdt.fit(features, lables)

gbdt\_model\_path = "D:/~STUDY~/Grade3/信息系统设计/final\_files/data/模型/gbdt\_6.model"

joblib.dump(gbdt, gbdt\_model\_path)

2.损失函数

#损失函数：偏差=|真实值-预测值|/真实值

def error(true\_labels, predict\_labels):

deviation = abs(true\_labels - predict\_labels) / true\_labels

return deviation.mean()

3.测试集测试

#将12.25~12.31的数据当作测试集测试

line6\_passenger\_hour\_test\_path = "D:/~STUDY~/Grade3/信息系统设计/final\_files/data/过程数据/line6\_passenger\_hour\_test.csv"

line\_passenger\_hour\_test = pd.read\_csv(line6\_passenger\_hour\_test\_path)

test\_labels = line\_passenger\_hour\_test['card\_id'] #得到测试集标签

gbdt\_6\_model\_path = "D:/~STUDY~/Grade3/信息系统设计/final\_files/data/模型/gbdt\_6.model"

gbdt\_model = joblib.load(gbdt\_6\_model\_path) #载入模型

test\_data\_path = "D:/~STUDY~/Grade3/信息系统设计/final\_files/data/过程数据/test\_line6\_weather\_date\_final.csv"

test\_data = pd.read\_csv(test\_data\_path)

features =test\_data #得到测试集特征

gbdt\_predict\_labels = gbdt\_model.predict(features) #预测

plt.plot(gbdt\_predict\_labels,label=u'predict\_value')

plt.plot(test\_labels,label=u'test\_value')

plt.xlabel("No.") #xlabel、ylabel：分别设置X、Y轴的标题文字

plt.ylabel("passenger crowding")

plt.legend(loc=0,ncol=2)

plt.show()

err = error(gbdt\_predict\_labels, test\_labels) #得出损失函数

print('错误率为:'+str(err\*100)+'%')

模型错误率如图5-11所示。

4.自定义特征并预测

#输入特征

test\_data1=pd.DataFrame({'temperature\_average':[20],

'wind\_average':[0],

'weather\_average':[6],

'time':[12],

'isholiday':[0],

'dayofweek':[1]})

features =test\_data1

gbdt\_predict\_labels = gbdt\_model.predict(features) #预测

print(gbdt\_predict\_labels)

5.3.3 百度地图API调用

1.申请密钥

2.地址编码服务

相关操作如下：

#调用地址编码服务获取经纬度 getjingwei.py

import requests

import json

url = 'http://api.map.baidu.com/geocoding/v3/'

params = { 'address' : '北京欢乐谷-极速飞车', #输入地点

'ak' : 'YdBcaxxxxxxxxxxxxxxxxxxxxxG', #百度密钥

'output': 'json' #输出结果设置为json格式

}

res = requests.get(url,params)

jd = json.loads(res.text) #将json格式转化为Python字典

print(jd)

coords = jd['result']['location']

print(coords)

3.轻量级路线规划服务

相关操作如下：

(2)获取任意两点之间的步行时间，写入文档。

#调用轻量级路线规划服务获取步行时间getdistance.py

import pandas

import csv

import json

from urllib.request import urlopen

#原数据文件格式：序号+起点纬度+起点经度+终点纬度+终点经度

origin\_path = 'D:/~STUDY~/Grade3/信息系统设计/final\_files/data/过程数据/起点终点坐标.xlsx' #原始坐标文件路径

result\_path = r'D:/~STUDY~/Grade3/信息系统设计/final\_files/data/过程数据/walk\_result\_raw.csv' #爬取数据文件保存路径

#声明坐标格式,bd09ll(百度经纬度坐标);bd09mc(百度摩卡托坐标);gcj02(国测局加密坐标),wgs84(gps设备获取的坐标)

cod = r"&coord\_type=bd09ll"

#AK为从百度地图网站申请的秘钥

AK = ['YdBcxxxxxxxxxxxxxxxxxxxG',]

dfBase = pandas.read\_excel(origin\_path, names=['序号','起点纬度','起点经度','终点纬度','终点经度'])

dfBase.head()

dataList = [] #储存获取的路线数据

akn = 0 #使用第几个ak

for i in range(len(dfBase)):

print(i)

ak = AK[akn]

out\_lat = dfBase.at[i,'起点纬度']

out\_lng = dfBase.at[i,'起点经度']

des\_lat = dfBase.at[i,'终点纬度']

des\_lng = dfBase.at[i,'终点经度']

#获取步行路径url\_walk=r"http://api.map.baidu.com/routematrix/v2/walking?output=json&origins={0},{1}&destinations={2},{3}&{4}&tactics=11&ak={4}".format(out\_lat,out\_lng,des\_lat,des\_lng,ak)

result\_walk = json.loads(urlopen(url\_walk).read()) #json转dict

status\_walk = result\_walk['status']

print('ak秘钥：{0} 获取步行路线状态码status：{1}'.format(ak, status\_walk))

if status\_walk == 0: #状态码为0：无异常

timesec\_walk = result\_walk['result'][0]['duration']['value']#耗时(秒)

elif status\_walk == 302 or status\_walk == 210 or status\_walk == 201:

#302:额度不足;210:IP验证未通过

timesec\_walk = 'AK错误'

akn += 1

ak = AK[akn]

else:

timesec\_walk = '请求错误'

dataList.append([ak,status\_walk,timesec\_walk])

dfAll = pandas.DataFrame(dataList, columns=['ak','status\_walk','timesec\_walk'])

dfAll.to\_csv(result\_path) #将生成的cvs保存到路径

walk\_time\_result.csv文件结果，如图5-17所示。

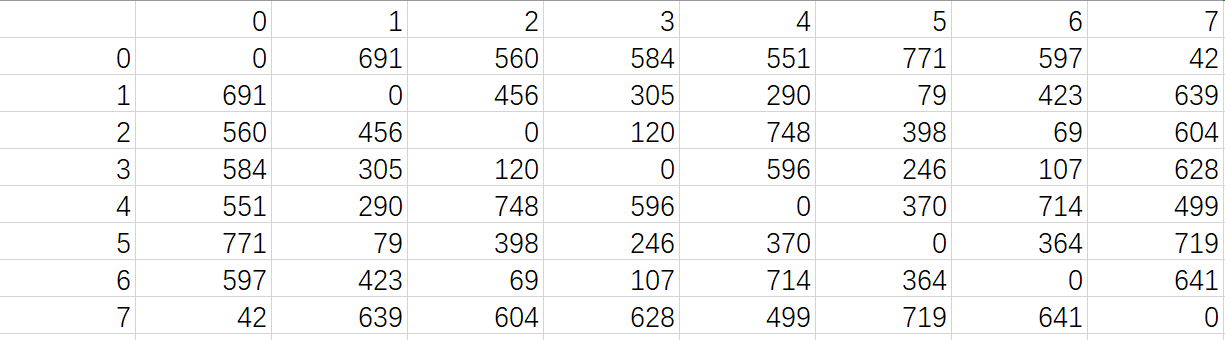


图5-17 walk\_time\_result.csv结果

5.3.4 GUI界面设计

1.手绘地图导入

相关代码如下：

#背景初始化部分

window = tkinter.Tk()

window.geometry('900x500')

window.title('智能导航系统——欢乐谷')

#显示图片

#通过PIL打开图片

img = Image.open('C:/Users/99509/Desktop/map.jpg')

img = img.resize((750,500),Image.ANTIALIAS) #Image.ANTIALIAS使图片不模糊

#通过PIL生成PhotoImage对象，即可正常加载

photo = ImageTk.PhotoImage(img)

imageLabel = Label(window, image=photo)

imageLabel.pack(side=LEFT)

2.下拉菜单设计

相关代码如下：

ss=Label(window,text="您当前位置为",justify=RIGHT)

ss.pack()

comvalue=tkinter.StringVar() #窗体自带的文本，新建一个值

comboxlist=ttk.Combobox(window,textvariable=comvalue) #初始化

comboxlist["values"]=("入口","玛雅天灾","雪域金翅","异域魔窟","奥德赛之旅","太阳神车","天地双雄","能量风暴")

all\_paths\_first=""

def xFunc(event):

global all\_paths\_first

#print(comboxlist.get())

#当前位置

if comboxlist.get()=="入口":

all\_paths\_first=0

elif comboxlist.get()=="玛雅天灾":

all\_paths\_first=1

elif comboxlist.get()=="雪域金翅":

all\_paths\_first=2

elif comboxlist.get()=="异域魔窟":

all\_paths\_first=3

elif comboxlist.get()=="奥德赛之旅":

all\_paths\_first=4

elif comboxlist.get()=="太阳神车":

all\_paths\_first=5

elif comboxlist.get()=="天地双雄":

all\_paths\_first=6

elif comboxlist.get()=="能量风暴":

all\_paths\_first=7

comboxlist.bind("<<ComboboxSelected>>",xFunc)

#绑定事件,(下拉列表框被选中时，绑定xFunc()函数)

comboxlist.pack()

3.复选框设计

#count判断是否需要调用百度地图API,奇数表示选中

count1=0

N = []

def myEvent1():

global count1

if count1%2==0:

count1+=1

else:

count1+=1

#项目选择部分

v1=IntVar()

c1=Checkbutton(window,text='入口',variable=v1,justify=RIGHT,command=myEvent1) #存放选中状态

c1.pack()

l1=Label(window,textvariable=v1,justify=RIGHT)

l1.pack() #未选中显示为0，选中显示为1

4.最短路径结果输出界面设计

相关代码如下：

total=[] #存放被选择的目的地

total\_n=[] #存放未被选择的地点

#选择地点完成后调出最短路径结果输出界面

def create():

window2 = tkinter.Toplevel() #新建子窗口windows2

window2.geometry('300x200')

window2.title('计算页面')

s0=Label(window2,text="您选择了：") #文字框s0

s0.pack()

#选择目的地

if count1%2==1: #如果被选中

N.append(0) #N列表中增加0

total.append("入口") #total列表中增加“入口”

else: #如果未被选中

N\_notchoose.append(0) #N\_notchoose列表中增加0

total\_n.append("入口") #total\_n列表中增加“入口”

if count2%2==1: #以下同理

N.append(1)

total.append("玛雅天灾")

else:

N\_notchoose.append(1)

total\_n.append("玛雅天灾")

if count3%2==1:

N.append(2)

total.append("雪域金翅")

else:

N\_notchoose.append(2)

total\_n.append("雪域金翅")

if count4%2==1:

N.append(3)

total.append("异域魔窟")

else:

N\_notchoose.append(3)

total\_n.append("异域魔窟")

if count5%2==1:

N.append(4)

total.append("奥德赛之旅")

else:

N\_notchoose.append(4)

total\_n.append("奥德赛之旅")

if count6%2==1:

N.append(5)

total.append("太阳神车")

else:

N\_notchoose.append(5)

total\_n.append("太阳神车")

if count7%2==1:

N.append(6)

total.append("天地双雄")

else:

N\_notchoose.append(6)

total\_n.append("天地双雄")

if count8%2==1:

N.append(7)

total.append("能量风暴")

else:

N\_notchoose.append(7)

total\_n.append("能量风暴")

s\_total=Label(window2,text=total) #输出用户选中的地点

s\_total.pack()

s9=Label(window2,text="最佳路线为：") #文字框s9

s9.pack()

get\_time(N) #调用get\_time进行运算

PLAN=[] #用于存放计划路径

#输出路径

for i in range(0,len(N)+1):

if PATH[i]==0:

PLAN.append("入口")

elif PATH[i]==1:

PLAN.append("玛雅天灾")

elif PATH[i]==2:

PLAN.append("雪域金翅")

elif PATH[i]==3:

PLAN.append("异域魔窟")

elif PATH[i]==4:

PLAN.append("奥德赛之旅")

elif PATH[i]==5:

PLAN.append("太阳神车")

elif PATH[i]==6:

PLAN.append("天地双雄")

else:

PLAN.append("能量风暴")

s10=Label(window2,text=PLAN) #输出计划的最短路径

s10.pack()

s11=Label(window2,text="\n大约耗时："+str(round(TIME[0],2))+"小时")

#输出对应总耗时

s11.pack()

b2=Button(window2,text='猜你喜欢',command=create\_guess)

#新建“猜你喜欢”按钮进入智能推荐模块

b2.pack()

b3=Button(window2,text='退出',command=window.destroy) #退出按钮

b3.pack()

5.智能推荐结果输出设计

相关代码如下：

#创建智能推荐的页面

def create\_guess():

window3 = tkinter.Toplevel() #新建子窗口windows3

window3.geometry('400x250')

window3.title('猜你喜欢')

s0=Label(window3,text="您选择了：") #文字框s0

s0.pack()

s\_total=Label(window3,text=total) #输出用户选中的地点

s\_total.pack()

s7=Label(window3,text="为您推荐：") #文字框s7

s7.pack()

try:

guess\_time(N) #调用guess\_time函数，计算推荐地点及推荐最短路径

except:

tkinter.messagebox.showwarning("提示", "选取地点有误！\n请退出重新选取")

#加入异常处理,跳出提示框

PLAN=[] #存放推荐最短路径

#输出路径

for i in range(0,len(N)+2):

if PATH\_guess[i]==0:

PLAN.append("入口")

elif PATH\_guess[i]==1:

PLAN.append("玛雅天灾")

elif PATH\_guess[i]==2:

PLAN.append("雪域金翅")

elif PATH\_guess[i]==3:

PLAN.append("异域魔窟")

elif PATH\_guess[i]==4:

PLAN.append("奥德赛之旅")

elif PATH\_guess[i]==5:

PLAN.append("太阳神车")

elif PATH\_guess[i]==6:

PLAN.append("天地双雄")

else:

PLAN.append("能量风暴")

s8=Label(window3,text=location[recommend]) #输出推荐的地点

s8.pack()

s9=Label(window3,text="加入推荐地点的最佳路线为：") #文本框s9

s9.pack()

s10=Label(window3,text=PLAN) #输出推荐的最短路径

s10.pack()

s11=Label(window3,text="\n大约耗时："+str(round(TIME\_guess[0],2))+"小时") #输出推荐的总耗时

s11.pack()

s12=Label(window3,text="预计比原路线多花费："+str(round((TIME\_guess[0]-TIME[0]),2))+"小时") #输出推荐一个地点后的总耗时与之前总耗时的差

s12.pack()

b3=Button(window3,text='返回',command=window3.destroy)

#“返回”按钮，返回上一界面

b3.pack()

b4=Button(window3,text='退出',command=window.destroy)

#“退出”按钮，退出程序

b4.pack()

5.3.5 路径规划

相关代码如下：

#获得最佳路径和出游时间

def get\_time(N):

global TIME

global PATH

gbdt\_6\_model\_path = "D:/~STUDY~/Grade3/信息系统设计/final\_files/data/模型/gbdt\_6.model"

gbdt\_model = joblib.load(gbdt\_6\_model\_path) #加载模型

#初始化客流模型顺序不可改变

features=pd.DataFrame({'temperature\_average':[2],

'wind\_average':[0],

'weather\_average':[6],

'time':[0],

'isholiday':[0],

'dayofweek':[1]})

#N = [0,1,2,3] #GUI界面单击选择

n=len(N)+1

print(all\_paths\_first)

all\_paths\_tuple=list(itertools.permutations(N,n-1))

#得到的全排列是元组tuple

all\_paths=[]

#tuple类型不能插入操作，所以转换成list

for i in range(0,math.factorial(n-1)):

all\_paths.append(list(all\_paths\_tuple[i]))

all\_paths[i].insert(0,all\_paths\_first)

path = list()

all\_time = float('inf') #无穷大

#第i条路线

for i in range(0,math.factorial(n-1)):

time = 0 #(h)

nowtime = 9 #可以获取当前时间，需要事件表示的转换(h)

print("第"+str(i+1)+"条路线为："+str(all\_paths[i]))

#第i条路线的第j个地点

for j in range(0,n-1):

time = time + (walk\_time[all\_paths[i][j]][all\_paths[i][j+1]]/3600)

#到达一个地点的时间

nowtime = nowtime+time

features['time'] = nowtime

gbdt\_predict\_labels = gbdt\_model.predict(features)/2000

#客流量/2000当作时间（h）

print("现在的时间是："+str(nowtime)+" 此时"+str(j)+"点的排队时间为："+str(gbdt\_predict\_labels)+"h")

time = time + gbdt\_predict\_labels + (PLAYTIME[j]/60)

print("耗时为："+str(time))

print("===================================================================")

if all\_time>time:

all\_time = time

path = all\_paths[i]

print("耗时最短的路径为："+str(path)+" 耗时为："+str(all\_time))

PATH = path

TIME = all\_time

5.3.6 智能推荐

相关代码如下：

#智能推荐“猜你喜欢”的运算

def guess\_time(N):

global TIME\_guess

global PATH\_guess

global recommend

path\_whole = list()

all\_time\_whole = float('inf') #无穷大

gbdt\_6\_model\_path = "D:/~STUDY~/Grade3/信息系统设计/final\_files/data/模型/gbdt\_6.model"

gbdt\_model = joblib.load(gbdt\_6\_model\_path) #加载模型

features=pd.DataFrame({'temperature\_average':[2],

'wind\_average':[0],

'weather\_average':[6],

'time':[0],

'isholiday':[0],

'dayofweek':[1]})

#N = [0,1,2,3] #GUI界面单击选择

print('起点是：'+str(all\_paths\_first))

for nn in range(0,len(N\_notchoose)):

#把未选择的地点分别加入到已选择的队列中进行计算

guess\_path = []

guess\_path = N[:]

guess\_path.append(N\_notchoose[nn]) #构建推荐的列表

print(guess\_path)

if(all\_paths\_first in guess\_path):

pass #如果未选择的地点是起点，则跳过

else:

all\_paths\_tuple=list(itertools.permutations(guess\_path,len(guess\_path)))

#得到的全排列是元组tuple

all\_paths=[]

#tuple类型不能插入操作，所以转换成list

#for i in range(0,math.factorial(n-1)):调试代码

for i in range(0,math.factorial(len(guess\_path))):

all\_paths.append(list(all\_paths\_tuple[i]))

all\_paths[i].insert(0,all\_paths\_first)

path = list()

all\_time = float('inf') #无穷大

#第i条路线

for i in range(0,math.factorial(len(guess\_path))):

time = 0 #(h)

nowtime = 9 #可以获取当前时间，需要事件表示的转换(h)

print("第"+str(i+1)+"条路线为："+str(all\_paths[i]))

#第i条路线的第j个地点

for j in range(0,len(guess\_path)):

time = time + (walk\_time[all\_paths[i][j]][all\_paths[i][j+1]]/3600)

#到达一个地点的时间

nowtime = nowtime+time

features['time'] = nowtime

gbdt\_predict\_labels = gbdt\_model.predict(features)/2000

#客流量/2000当作时间（h）

print("现在的时间是："+str(nowtime)+" 此时"+str(j)+"点的排队时间为："+str(gbdt\_predict\_labels)+"h")

time = time + gbdt\_predict\_labels + (PLAYTIME[j]/60)

print("耗时为："+str(time))

print("")

if all\_time>time:

all\_time = time

path = all\_paths[i]

if all\_time\_whole>all\_time:

all\_time\_whole = all\_time

path\_whole=path[:]

recommend = N\_notchoose[nn]

print("推荐第"+str(recommend)+"个景点")

print("推荐第"+str(recommend)+"个景点")

print("智能推荐耗时最短的路径为a："+str(path\_whole)+" 耗时为："+str(all\_time\_whole))

PATH\_guess = path\_whole

TIME\_guess = all\_time\_whole

在cmd窗口输出所有路径及该路径每到一节点的时间，如图5-22所示。

项目6 基于人工智能的面相推荐分析

6.3 模块实现

6.3.1 数据预处理

下载数据集analysis的相关代码如下：

from \_\_future\_\_ import with\_statement  
from google\_images\_download import google\_images\_download  
import json, os  
TRAIN\_DATA\_DIR\_PATH = "train\_imgs"  
 #下载analysis用于区域分类  
with open('data/analysis.json',encoding='utf-8') as f:  
 analysis = json.load(f)  
global\_args = {  
 "limit":75,  
 "output\_directory":TRAIN\_DATA\_DIR\_PATH,  
 "prefix":"",  
 "keywords":"",  
 "prefix\_keywords":"面相"  
}  
 #从google\_images\_download中下载使用数据  
for region in analysis["face\_regions"]:   
 region\_name = region["name"]  
 for feature in region["features"]:  
 download\_args = global\_args  
 download\_args["output\_directory"] = os.path.join(TRAIN\_DATA\_DIR\_PATH, region\_name)  
 download\_args["keywords"] = feature["name"]  
 response = google\_images\_download.googleimagesdownload()  
 response.download(download\_args)  
 default\_fking\_ugly\_dirname = os.path.join(TRAIN\_DATA\_DIR\_PATH, region\_name, download\_args["prefix\_keywords"] + " " + feature["name"])  
 os.rename(default\_fking\_ugly\_dirname,default\_fking\_ugly\_dirname.replace( download\_args["prefix\_keywords"] + " ", ""))

6.3.2 模型构建

1.定义模型结构

2.交叉验证模型优化

交叉验证代码如下：

scores = cross\_val\_score(svms[region\_name.encode()], X, y, cv=5)

6.3.3 模型训练及保存

#SVM分类  
import sys  
path = 'E:\Anaconda\envs\\tensorflow36\Lib\site-packages\libsvm\python'  
sys.path.append(path)  
from svmutil import \*  
from utils import \*  
from sklearn import svm  
from sklearn.externals import joblib  
from sklearn.model\_selection import train\_test\_split  
from sklearn.model\_selection import cross\_val\_score  
import numpy as np  
#设置数据模型存储路径  
SAVE\_PATH="data/trained\_svms.pkl"  
SAVE\_TRAIN\_DATA\_PATH = "data/train\_data.pkl"  
LIBSVM\_SVMS\_PATH = "data/%s.svm"  
LIBSVM\_LABELS\_PATH = "data/labels.txt"  
GET\_CROSS\_VAL = False #是否进行交叉验证  
IS\_BUILD\_LIBSVM\_MODEL = False #判断是否是LIBSVM模型  
#加载模型数据  
if os.path.isfile(SAVE\_TRAIN\_DATA\_PATH):  
 data = joblib.load(SAVE\_TRAIN\_DATA\_PATH)  
else:  
 data = loadData()  
 joblib.dump(data, SAVE\_TRAIN\_DATA\_PATH) #将模型保存至本地  
svms = {}  
if IS\_BUILD\_LIBSVM\_MODEL:  
 labels\_file = open(LIBSVM\_LABELS\_PATH, 'w')  
for region\_name, features in data.items(): #训练数据集  
 print("training svm for %s"% (region\_name))  
 #将数据分为训练集合和测试集  
 if not IS\_BUILD\_LIBSVM\_MODEL:  
 X = []   
 y = []  
 for feature\_name, feature\_shapes in features.items(): #脸部特征提取  
 for shape in feature\_shapes:  
 X.append(shape.flatten()) #记录向量特征  
 y.append(feature\_name) #记录向量名称  
 X = np.squeeze(np.array(X)) #改变向量维数  
 y = np.array(y,dtype='S128') #128位字符串  
 #分割数据  
 #X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y)  
 svms[region\_name.encode()] = svm.SVC(kernel="linear", probability=True) #设置支持向量机参数  
 if GET\_CROSS\_VAL:#交叉验证  
 scores = cross\_val\_score(svms[region\_name.encode()], X, y, cv=5)

#cv表示选择折数  
 print("Cross val score: ", scores)  
 print("Accuracy:%0.2f(+/- %0.2f)"%(scores.mean(),scores.std()\* 2))  
 #训练部署  
 svms[region\_name.encode()].fit(X, y)#用训练数据拟合分类器模型  
 else:#对于LIBSVM的模型处理  
 #为方便程序重复使用模型，运算效率更高，将模型保存为.svm格式

X = []   
 y = []  
 for i, (feature\_name, feature\_shapes) in enumerate(features.items()):

#遍历脸部特征，以下同上  
 for shape in feature\_shapes:  
 X.append(shape.flatten())  
 y.append(i)  
 X = np.squeeze(np.array(X))  
 y = np.array(y,dtype='uint8')#这里改为使用int8存储  
 #将LIBSVM模型写入文件  
 labels\_file.write("%s\n" % region\_name)  
 labels\_file.write(LIBSVM\_SVMS\_PATH % region\_name)  
 labels\_file.write(" ")  
 labels\_file.write(" ".join([k.decode() for k in features.keys()]))  
 labels\_file.write("\n")  
 #将数据训练并保存  
 prob = svm\_problem(y.tolist(), X.tolist()) #tolist使数据列表化  
 param = svm\_parameter("-h 0 -s 0 -t 1 -b 1")  
 m=svm\_train(prob, param)  
 svm\_save\_model(LIBSVM\_SVMS\_PATH % region\_name, m)  
if IS\_BUILD\_LIBSVM\_MODEL:  
 labels\_file.close()  
print("training svm... Done")   
joblib.dump(svms, SAVE\_PATH) #保存模型  
print("svm saved!")

模型被保存后，可以被重用，也可以移植到其他环境中使用。

6.3.4 模型应用

1.摄像头调用

相关代码如下：

def getImgFromCam():  
 vs = VideoStream(usePiCamera=False).start() #调用计算机摄像头  
 time.sleep(2.0)  
 while True:  
 frame = vs.read()  
 frame = imutils.resize(frame, width=400)  
 gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)  
 #在灰度框中检测人脸  
 rects = detector(gray, 0)  
 if rects is not None and len(rects) > 0: #当检测到有人脸存在时结束  
 return frame

2.模型导入及调用

将训练好的.svm文件放入data目录下，并声明模型存放路径。

SAVE\_PATH="data/trained\_svms.pkl"  
SAVE\_TRAIN\_DATA\_PATH = "data/train\_data.pkl"  
LIBSVM\_SVMS\_PATH = "data/%s.svm"  
LIBSVM\_LABELS\_PATH = "data/labels.txt"

#将LIBSVM模型写入文件

labels\_file.write("%s\n" % region\_name)  
labels\_file.write(LIBSVM\_SVMS\_PATH % region\_name)  
labels\_file.write(" ")  
labels\_file.write(" ".join([k.decode() for k in features.keys()]))  
labels\_file.write("\n")

3.前端代码

##### 1)UI设计

相关代码如下：

import sys  
from PyQt5 import QtWidgets  
from untitled import \*  
from PyQt5.QtWidgets import QFileDialog  
from try\_svm import \*  
from facereading import \*  
import dlib #人脸处理库Dlib  
import numpy as np #数据处理库Numpy  
import cv2 #图像处理库OpenCV  
import os #读写文件  
import shutil #读写文件  
from QCandyUi.CandyWindow import colorful  
global imgName  
imgname = 00000  
from utils import \*  
class MyPyQT\_Form(QtWidgets.QMainWindow,Ui\_MainWindow):  
 def \_\_init\_\_(self):  
 super(MyPyQT\_Form,self).\_\_init\_\_()  
 self.setupUi(self)  
 #实现pushButton\_click()函数，textEdit是文本框的ID  
 def slot1(self):  
 detector = dlib.get\_frontal\_face\_detector()  
 #OpenCV调用摄像头  
 cap = cv2.VideoCapture(0)  
 #人脸截图的计数器  
 cnt\_ss = 0  
 #存储人脸的文件夹  
 current\_face\_dir = ""  
 #保存人脸图像的路径  
 path\_photos\_from\_camera = "data/data\_faces\_from\_camera/"  
 #新建保存人脸图像文件和数据CSV文件夹  
 def pre\_work\_mkdir():  
 #新建文件夹  
 if os.path.isdir(path\_photos\_from\_camera):  
 pass  
 else:  
 os.mkdir(path\_photos\_from\_camera)  
 pre\_work\_mkdir()  
 #可选, 默认关闭  
 #删除之前存的人脸数据文件夹   
 def pre\_work\_del\_old\_face\_folders():  
 folders\_rd = os.listdir(path\_photos\_from\_camera)  
 for i in range(len(folders\_rd)):  
 shutil.rmtree(path\_photos\_from\_camera + folders\_rd[i])  
 if os.path.isfile("data/features\_all.csv"):  
 os.remove("data/features\_all.csv")  
 #每次程序录入时删掉之前存的人脸数据  
 #如果打开，每次进行人脸录入时都会删掉之前的人脸图像文件夹 person\_1/,person\_2/,person\_3/...  
 #如果启用此功能，将删除目录中所有旧数据person\_1/,person\_2/,/person\_3/...  
 #pre\_work\_del\_old\_face\_folders()  
 #Check people order: person\_cnt  
 #如果有之前录入的人脸  
 #在之前person\_x的序号按照person\_x+1开始录入  
 if os.listdir("data/data\_faces\_from\_camera/"):  
 #获取已录入的最后一个人脸序号  
 person\_list = os.listdir("data/data\_faces\_from\_camera/")  
 person\_num\_list = []  
 for person in person\_list:  
 person\_num\_list.append(int(person.split('\_')[-1]))  
 person\_cnt = max(person\_num\_list)  
 #如果第一次存储或者没有之前录入的人脸,按照person\_1开始录入  
 else:  
 person\_cnt = 0  
 #flag用来控制是否保存图像  
 save\_flag = 1  
 #flag用来检查是否先按n再按s  
 press\_n\_flag = 0  
 while cap.isOpened():  
 flag, img\_rd = cap.read()  
 #print(img\_rd.shape)调试代码  
 #默认Windows和Ubuntu为480\*640，macOS为1280\*720  
 kk = cv2.waitKey(1)  
 img\_gray = cv2.cvtColor(img\_rd, cv2.COLOR\_RGB2GRAY)  
 #人脸  
 faces = detector(img\_gray, 0)  
 #要写的字体  
 font = cv2.FONT\_ITALIC  
 #按下n新建存储人脸的文件夹  
 if kk == ord('n'):  
 person\_cnt += 1  
 current\_face\_dir = path\_photos\_from\_camera + "person\_" + str(person\_cnt)  
 os.makedirs(current\_face\_dir)  
 print('\n')  
 print("新建的人脸文件夹 / Create folders: ", current\_face\_dir)  
 cnt\_ss = 0 #将人脸计数器清零  
 press\_n\_flag = 1 #已经按下n  
 #检测到人脸  
 if len(faces) != 0:  
 #矩形框  
 for k, d in enumerate(faces):  
 #(x,y), (宽度width, 高度height)  
 pos\_start = tuple([d.left(), d.top()])  
 pos\_end = tuple([d.right(), d.bottom()])  
 #计算矩形框大小  
 height = (d.bottom() - d.top())  
 width = (d.right() - d.left())  
 hh = int(height / 2)  
 ww = int(width / 2)  
 #设置颜色  
 color\_rectangle = (255, 255, 255)  
 #判断人脸矩形框是否超出640\*480  
 if (d.right() + ww) > 640 or (d.bottom() + hh > 480) or (d.left() - ww < 0) or (d.top() - hh < 0):  
 cv2.putText(img\_rd, "OUT OF RANGE", (20, 300), font, 0.8, (0, 0, 255), 1, cv2.LINE\_AA)  
 color\_rectangle = (0, 0, 255)  
 save\_flag = 0  
 if kk == ord('s'):  
 print("请调整位置/Please adjust your position")  
 else:  
 color\_rectangle = (255, 255, 255)  
 save\_flag = 1  
 cv2.rectangle(img\_rd,  
 tuple([d.left() - ww, d.top() - hh]),  
 tuple([d.right() + ww, d.bottom() + hh]),  
 color\_rectangle, 2)  
 #根据人脸大小生成空的图像  
 im\_blank = np.zeros((int(height \* 2), width \* 2, 3), np.uint8)  
 if save\_flag:  
 #按下s保存摄像头中的人脸到本地  
 if kk == ord('s'):  
 #检查是否先按n新建文件夹  
 if press\_n\_flag:  
 cnt\_ss += 1  
 for ii in range(height \* 2):  
 for jj in range(width \* 2):  
 im\_blank[ii][jj] = img\_rd[d.top() - hh + ii][d.left() - ww + jj]  
 cv2.imwrite(current\_face\_dir + "/img\_face\_" + str(cnt\_ss) + ".jpg", im\_blank)  
 print("写入本地 / Save into：", str(current\_face\_dir) + "/img\_face\_" + str(cnt\_ss) + ".jpg")  
 else:  
 print("请在按 'S' 之前先按 'N' 来建文件夹 / Please press 'N' before 'S'")  
 #显示人脸数  
 cv2.putText(img\_rd, "Faces: " + str(len(faces)), (20, 100), font, 0.8, (0, 255, 0), 1, cv2.LINE\_AA)  
 #添加说明  
 cv2.putText(img\_rd, "Face Register", (20, 40), font, 1, (0, 0, 0), 1, cv2.LINE\_AA)  
 cv2.putText(img\_rd, "N: Create face folder", (20, 350), font, 0.8, (0, 0, 0), 1, cv2.LINE\_AA)  
 cv2.putText(img\_rd, "S: Save current face", (20, 400), font, 0.8, (0, 0, 0), 1, cv2.LINE\_AA)  
 cv2.putText(img\_rd, "Q: Quit", (20, 450), font, 0.8, (0, 0, 0), 1, cv2.LINE\_AA)  
 #按下q键退出  
 located = str('D:\\pylearn\\Face-Reading\\')+str(str(current\_face\_dir) + "\\img\_face\_" + str(cnt\_ss) + ".jpg")  
 load\_face = cv2.imread(located)  
 if kk == ord('q'):  
 print(located)  
 png = QtGui.QPixmap(located).scaled(self.label.width(), self.label.height()) #适应设计标签时的大小  
 self.label.setPixmap(png)  
 wenben = apply(load\_face)  
 while not wenben.empty():  
 temp = wenben.get()  
 #print(temp)调试代码  
 self.textEdit.append(temp)  
 break  
 #如果需要摄像头窗口大小可调  
 #cv2.namedWindow("camera", 0)调试代码  
 cv2.imshow("camera", img\_rd)  
 #释放摄像头  
 cap.release()  
 cv2.destroyAllWindows()  
 def duqu(self):  
 global imgName  
 print("笑一笑就好")  
 imgName, imgType = QFileDialog.getOpenFileName(self,  
 "打开图片",  
 "", " \*.jpg;;\*.png;;\*.jpeg;;\*.bmp;;All Files (\*)")  
 #显示图片  
 #print(str(imgName))调试代码  
 png = QtGui.QPixmap(imgName).scaled(self.label.width(), self.label.height()) #适应设计标签时的大小  
 self.label.setPixmap(png)  
 def suanming(self):  
 self.textEdit.setReadOnly(True)  
 img2 = cv2.imread(imgName)  
 wenben = apply(img2)  
 while not wenben.empty():  
 temp = wenben.get()  
 #print(temp)  
 self.textEdit.append(temp)  
if \_\_name\_\_ == '\_\_main\_\_':  
 app = QtWidgets.QApplication(sys.argv)  
 my\_pyqt\_form = MyPyQT\_Form()  
 my\_pyqt\_form.show()  
 sys.exit(app.exec\_())

##### 2)训练函数

相关代码如下：

#SVM分类  
import sys  
path = 'E:\Anaconda\envs\\tensorflow36\Lib\site-packages\libsvm\python'  
sys.path.append(path)  
from svmutil import \*  
from utils import \*  
from sklearn import svm  
from sklearn.externals import joblib  
from sklearn.model\_selection import train\_test\_split  
from sklearn.model\_selection import cross\_val\_score  
import numpy as np  
#设置数据模型存储路径  
SAVE\_PATH="data/trained\_svms.pkl"  
SAVE\_TRAIN\_DATA\_PATH = "data/train\_data.pkl"  
LIBSVM\_SVMS\_PATH = "data/%s.svm"  
LIBSVM\_LABELS\_PATH = "data/labels.txt"  
GET\_CROSS\_VAL = False #是否进行交叉验证  
IS\_BUILD\_LIBSVM\_MODEL = False #判断是否是LIBSVM模型  
#加载模型数据  
if os.path.isfile(SAVE\_TRAIN\_DATA\_PATH):  
 data = joblib.load(SAVE\_TRAIN\_DATA\_PATH)  
else:  
 data = loadData()  
 joblib.dump(data, SAVE\_TRAIN\_DATA\_PATH) #将模型保存至本地  
svms = {}  
if IS\_BUILD\_LIBSVM\_MODEL:  
 labels\_file = open(LIBSVM\_LABELS\_PATH, 'w')  
for region\_name, features in data.items(): #训练数据集  
 print("training svm for %s"% (region\_name))  
 #分割数据为训练集和测试集  
 if not IS\_BUILD\_LIBSVM\_MODEL:   
 X = []   
 y = []  
 for feature\_name, feature\_shapes in features.items(): #脸部特征提取  
 for shape in feature\_shapes:  
 X.append(shape.flatten()) #记录向量特征  
 y.append(feature\_name) #记录向量名称   
 X = np.squeeze(np.array(X)) #改变向量维数  
 y = np.array(y,dtype='S128') #128位字符串  
 #分割数据  
 #X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y)  
 svms[region\_name.encode()] = svm.SVC(kernel="linear", probability=True)

#设置支持向量机参数  
 if GET\_CROSS\_VAL: #交叉验证  
 scores = cross\_val\_score(svms[region\_name.encode()], X, y, cv=5)

#cv表示选择折数  
 print("Cross val score: ", scores)  
 print("Accuracy: %0.2f (+/-%0.2f)"%(scores.mean(),scores.std()\* 2))  
 svms[region\_name.encode()].fit(X, y) #用训练数据拟合分类器模型  
 print(svms[region\_name.encode()].score(X,y))  
 else: #对于LIBSVM模型的处理  
 X = []   
 y = []  
 for i, (feature\_name, feature\_shapes) in enumerate(features.items()):

#遍历脸部特征，以下同上  
 for shape in feature\_shapes:  
 X.append(shape.flatten())  
 y.append(i)   
 X = np.squeeze(np.array(X))  
 y = np.array(y,dtype='uint8') #这里使用int8存储  
 #将LIBSVM模型写入文件  
 labels\_file.write("%s\n" % region\_name)  
 labels\_file.write(LIBSVM\_SVMS\_PATH % region\_name)  
 labels\_file.write(" ")  
 labels\_file.write(" ".join([k.decode() for k in features.keys()]))  
 labels\_file.write("\n")  
 #将数据训练并保存  
 prob = svm\_problem(y.tolist(), X.tolist()) #tolist使数据列表化  
 param = svm\_parameter("-h 0 -s 0 -t 1 -b 1")  
 m=svm\_train(prob, param)  
 svm\_save\_model(LIBSVM\_SVMS\_PATH % region\_name, m)  
if IS\_BUILD\_LIBSVM\_MODEL:  
 labels\_file.close()   
print("training svm... Done")   
joblib.dump(svms, SAVE\_PATH) #保存模型  
print("svm saved!")

##### 3)自定义工具函数

相关代码如下：

from \_\_future\_\_ import print\_function  
import os, sys  
import cv2  
import dlib  
import imutils  
from imutils.video import VideoStream  
from imutils import face\_utils  
from imutils.face\_utils import FaceAligner  
from glob import glob  
import numpy as np  
#配置  
USE\_REGION = True # use part of the feature to train the svm, e.g. only use mouth feature points  
LANDMARK\_PATH = "data/shape\_predictor\_68\_face\_landmarks.dat"

#数据集Dlib人脸68个关键点  
#Dlib初始化配置  
detector = dlib.get\_frontal\_face\_detector()  
predictor = dlib.shape\_predictor(LANDMARK\_PATH)  
fa = FaceAligner(predictor, desiredFaceWidth=400)  
faceRegions = { #判断区域分类  
 "eye\_left": list(range(36,41+1)),  
 "eye\_right": list(range(42, 47+1)),  
 "nose": list(range(27, 35+1)),  
 "mouth": list(range(48, 60+1)),  
 "face": list(range(0, 16+1)),  
 "eyebrow\_left": list(range(17,21+1)),  
 "eyebrow\_right": list(range(22,26+1))  
}  
faceRegions["eyes"] = faceRegions["eye\_left"] + faceRegions["eye\_right"]  
faceRegions["eyebrows"] = faceRegions["eyebrow\_left"] + faceRegions["eyebrow\_right"]  
def loadData(dir="train\_imgs"):  
 data = {"face":{}, "eyebrows":{}, "eyes":{}, "nose":{}, "mouth":{}}  
 tc = 0  
 for region\_name, v in data.items():  
 paths = os.path.join(dir, region\_name, '\*/\*.\*') #路径设置  
 rc = 0  
 for path in glob(paths): #对于训练数据下的所有目录文件提取并进行训练  
 \_, feature\_name = os.path.split(os.path.dirname(path))  
 feature\_name = feature\_name.encode()  
 if feature\_name not in v: #若区域未设置该向量为空  
 v[feature\_name] = []  
 img = cv2.imread(path)  
 if img is None: #若图像为空则跳过  
 continue  
 points= getNormalizedFeature(region\_name, feature\_name, img)  
 #若没有检测到人脸则跳过  
 if points is not None:  
 v[feature\_name].append(points)  
 rc+=1  
 tc+=1  
 sys.stdout.write("\033[K")  
 print("loading...%s%d/%d"%(region\_name,rc,

tc), end="\r")  
 print("")   
 print("loading... Done")  
 return data  
#从图像中获取面部区域名称，面部区域特征的函数将图像进行标准化处理  
def getNormalizedFeature(region\_name, feature\_name, img):  
 img = imutils.resize(img, width=800)  
 gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)  
 rects = detector(gray, 0)  
 if len(rects) == 0: #没有检测到人脸  
 #sys.exit("No face is detected in %s of %s" % (feature\_name, region\_name))  
 return None  
 else: #面部特征处理并提取  
 faceImg = fa.align(img, gray, rects[0])  
 full\_rect = dlib.rectangle(0, 0, faceImg.shape[1], faceImg.shape[0])  
 shape = predictor(faceImg, full\_rect)  
 if USE\_REGION:  
 shape = face\_utils.shape\_to\_np(shape)[faceRegions[region\_name]]  
 else:  
 shape = face\_utils.shape\_to\_np(shape)  
 return shape  
#函数的重载，对仅提供图片参数的情况处理

def getNormalizedFeatures(img, display=False):  
 img = imutils.resize(img, width=800)  
 gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)  
 data = {"face":[], "eyebrows":[], "eyes":[], "nose":[], "mouth":[]}  
 rects = detector(gray, 0)  
 if len(rects) == 0: #没有检测到人脸  
 sys.exit("No face is detected")  
 return None  
 else: #图像特征提取  
 faceImg = fa.align(img, gray, rects[0])  
 full\_rect = dlib.rectangle(0, 0, faceImg.shape[1], faceImg.shape[0])  
 points = predictor(faceImg, full\_rect)  
 points = face\_utils.shape\_to\_np(points)  
 if display: #显示图像  
 cv2.imshow("face", faceImg)  
 cv2.waitKey()  
 for key in data: #关键特征提取  
 if USE\_REGION:  
 data[key] = points[faceRegions[key]]  
 else:  
 data[key] = points  
 return faceImg, data

##### 4)主活动类

相关代码如下：

#测试训练数据  
import argparse  
import json  
import time  
import queue  
from textwrap import fill  
import cv2  
import matplotlib.pyplot as plt  
import numpy as np  
from imutils.convenience import url\_to\_image  
from sklearn.externals import joblib  
from utils import \*  
#路径设置（图片保存和测试路径）  
TEST\_IMAGE\_PATH="test\_imgs\\test1.png"  
SAVE\_PATH="data/trained\_svms.pkl"  
#初始化选择系统，用于测试各项功能  
ap = argparse.ArgumentParser()  
ap.add\_argument("-c", "--camera", default=False, action="store\_true",  
 help="get input from camera")  
ap.add\_argument("-i", "--image", type=str, default=None,  
 help="input image")  
ap.add\_argument("-u", "--url", type=str, default=None,  
 help="input image url")  
args = vars(ap.parse\_args())  
#加载分析数据  
with open('data/analysis.json','rb') as f:  
 analysis = json.load(f)  
#核心算法  
def apply(img):  
 wenben = queue.Queue()  
 faceImg, data = getNormalizedFeatures(img, False)

#调用utils工具中的函数获取面部特征（眼、鼻、口、眉）  
 svms = joblib.load(SAVE\_PATH)#调用训练好的模型  
 #显示图像的测试函数  
 #plt.imshow(imutils.opencv2matplotlib(faceImg))调试代码  
 #plt.show()调试代码  
 for region\_name, points in data.items(): #图像data参数中的面部区域和特征点  
 X = [points.flatten()] #转变为向量形式处理  
 y = svms[region\_name.encode()].predict(X)[0].decode() #cv2当中的预测函数  
 prob = svms[region\_name.encode()].predict\_proba(X) #支撑向量机预测输出  
 max\_prob = np.amax(prob)\*100  
 wenben1 = "【 %s 】\t %s %f%%" % (region\_name, y, max\_prob)  
 for region in analysis["face\_regions"]: #文本存储判断的结果  
 if region["name"] == region\_name:  
 for feature in region["features"]:  
 if feature["name"] == y:  
 wenben2 = fill(feature["analysis"], width=18)  
 temp = str(wenben1) + '\n' + str(wenben2)  
 wenben.put(str(temp))  
 #print(wenben.get())  
 return wenben  
def getImgFromCam():  
 vs = VideoStream(usePiCamera=False).start() #调用摄像头  
 time.sleep(2.0)  
 while True:  
 frame = vs.read()  
 frame = imutils.resize(frame, width=400)  
 gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)  
 rects = detector(gray, 0)  
 if rects is not None and len(rects) > 0: #当检测到有人脸存在时结束  
 return frame  
if \_\_name\_\_ == '\_\_main\_\_':  
 if args["camera"]:  
 img = getImgFromCam()  
 elif args["image"] is not None:  
 img = cv2.imread(args["image"])  
 elif args["url"] is not None:  
 img = url\_to\_image(args["url"])  
 else:  
 img = cv2.imread(TEST\_IMAGE\_PATH)  
 apply(img)

项目7 图片情感分析与匹配音乐生成推荐

7.3 模块实现

7.3.1 数据预处理

1.图片部分

#-\*- coding: utf-8 -\*-

import sys

import csv

import traceback

import os

import pandas as pd

if sys.version\_info < (3, 0):

from urllib2 import urlopen

else:

from urllib.request import urlopen

import io

from colorthief import ColorThief

def get\_color(path):#获取图片的色彩信息

def rgb2hsv(tp):#将转换成hsv，h是色相，s是饱和度，v是明度

r,g,b=tp[0],tp[1],tp[2]

r, g, b = r/255.0, g/255.0, b/255.0

mx = max(r, g, b)

mn = min(r, g, b)

m = mx-mn

if mx == mn:

h = 0

elif mx == r:

if g >= b:

h = ((g-b)/m)\*60

else:

h = ((g-b)/m)\*60 + 360

elif mx == g:

h = ((b-r)/m)\*60 + 120

elif mx == b:

h = ((r-g)/m)\*60 + 240

if mx == 0:

s = 0

else:

s = m/mx

v = mx

h,s,v = round(h,3),round(s,3),round(v,3) #保留小数点后三位

h,s,v = str(h),str(s),str(v) #转成字符串类型能够写入csv

return h,s,v

fd = urlopen(path)

f = io.BytesIO(fd.read())

color\_thief = ColorThief(f) #调用colortheif()函数

mc=color\_thief.get\_color(quality=1)

#获取画面最主要颜色，不一定出现在画面中，是整体均值，一个元组（r,g,b）

cp=color\_thief.get\_palette(quality=1) #获取调色盘

hsv = rgb2hsv(mc)

color\_lists=[hsv] #用列表存储最主要颜色信息,[h,s,v]

clist=[color\_lists[0][0],color\_lists[0][1],color\_lists[0][2]]

#获取最主要颜色的h，s，v值

for c in cp:#遍历调色盘中的颜色

hp=rgb2hsv(c)

color\_lists.append(hp) #追加信息到列表中

for i in hp: #获取每个色彩的h，s，v值

clist.append(i)

return clist

def to\_csv(clist): #将色彩信息列表存储到.csv文件

try:

fpath = 'E:/college/synaes/image\_csv/2.csv'

#存储色彩信息的.csv文件地址

with open(fpath,'a',newline='')as f:

writer = csv.writer(f) #writer.writerow(["h0","s0","v0","h1","s1","v1","h2","s2","v2","h3","s3","v3","h4","s4","v4","h5","s5","v5","h6","s6","v6","h7","s7","v7","h8","s8","v8","h9","s9","v9","Label"])

writer.writerow(clist)

except:

print(traceback())

#get\_color\_to\_file("file:///E:/college/synaes/image\_test/test2.jpg")

path="E:/college/synaes/image/happy" #图片存储路径

file\_list=os.listdir(path)

for file in file\_list: #遍历路径中的每张图片

clist=get\_color("file:///"+path+"/"+file)

to\_csv(clist)

# -\*- coding: utf-8 -\*-

#从图像中抓取调色板

\_\_version\_\_ = '0.2.1'

import math

from PIL import Image #导入pillow中的image模块

class cached\_property(object):

#创建的装饰器将单一参数的方法转换为缓存实例的属性

def \_\_init\_\_(self, func):

self.func = func

def \_\_get\_\_(self, instance, type):

res = instance.\_\_dict\_\_[self.func.\_\_name\_\_] = self.func(instance)

return res

class ColorThief(object):

#抓取调色的类

def \_\_init\_\_(self, file):

self.image = Image.open(file) #打开图像

def get\_color(self, quality=10):

#获得主要的颜色，quality参数：1是最高的quality，数字越大，颜色返回越快

#返回tuple:(r, g, b)元组类型

palette = self.get\_palette(5, quality)

return palette[0]

def get\_palette(self, color\_count=10, quality=10):

#获得调色盘，用中值切割算法聚类相似颜色

#参数color\_count为调色盘的大小

#quality参数同上

#返回一个以(r,g,b)元组为元素的列表

image = self.image.convert('RGBA')

width, height = image.size

pixels = image.getdata() #像素

pixel\_count = width \* height

valid\_pixels = []

for i in range(0, pixel\_count, quality):

r, g, b, a = pixels[i]

#如果像素大部分是不透明的，而且不是白色的

if a >= 125:

if not (r > 250 and g > 250 and b > 250):

valid\_pixels.append((r, g, b))

#将数组传给quantize()函数，该函数对值进行聚类,使用中值切割算法

cmap = MMCQ.quantize(valid\_pixels, color\_count)

return cmap.palette

class MMCQ(object):

#MMCQ基本是Python端口(改进的中值切割量化)

#算法来自Leptonica库(http://www.leptonica.com/)

SIGBITS = 5

RSHIFT = 8 - SIGBITS

MAX\_ITERATION = 1000 #最大迭代次数

FRACT\_BY\_POPULATIONS = 0.75

@staticmethod

def get\_color\_index(r, g, b):

return (r << (2 \* MMCQ.SIGBITS)) + (g << MMCQ.SIGBITS) + b

@staticmethod

def get\_histo(pixels):

histo = dict()

for pixel in pixels:

rval = pixel[0] >> MMCQ.RSHIFT

gval = pixel[1] >> MMCQ.RSHIFT

bval = pixel[2] >> MMCQ.RSHIFT

index = MMCQ.get\_color\_index(rval, gval, bval)

histo[index] = histo.setdefault(index, 0) + 1

return histo

@staticmethod

def vbox\_from\_pixels(pixels, histo):

rmin = 1000000

rmax = 0

gmin = 1000000

gmax = 0

bmin = 1000000

bmax = 0

for pixel in pixels:

rval = pixel[0] >> MMCQ.RSHIFT

gval = pixel[1] >> MMCQ.RSHIFT

bval = pixel[2] >> MMCQ.RSHIFT

rmin = min(rval, rmin)

rmax = max(rval, rmax)

gmin = min(gval, gmin)

gmax = max(gval, gmax)

bmin = min(bval, bmin)

bmax = max(bval, bmax)

return VBox(rmin, rmax, gmin, gmax, bmin, bmax, histo)

@staticmethod

def median\_cut\_apply(histo, vbox): #中值切割

if not vbox.count:

return (None, None)

rw = vbox.r2 - vbox.r1 + 1

gw = vbox.g2 - vbox.g1 + 1

bw = vbox.b2 - vbox.b1 + 1

maxw = max([rw, gw, bw])

#如果只有一个像素，不进行切割

if vbox.count == 1:

return (vbox.copy, None)

#沿着选定的轴查找数组

total = 0

sum\_ = 0

partialsum = {}

lookaheadsum = {}

do\_cut\_color = None

if maxw == rw:

do\_cut\_color = 'r'

for i in range(vbox.r1, vbox.r2+1):

sum\_ = 0

for j in range(vbox.g1, vbox.g2+1):

for k in range(vbox.b1, vbox.b2+1):

index = MMCQ.get\_color\_index(i, j, k)

sum\_ += histo.get(index, 0)

total += sum\_

partialsum[i] = total

elif maxw == gw:

do\_cut\_color = 'g'

for i in range(vbox.g1, vbox.g2+1):

sum\_ = 0

for j in range(vbox.r1, vbox.r2+1):

for k in range(vbox.b1, vbox.b2+1):

index = MMCQ.get\_color\_index(j, i, k)

sum\_ += histo.get(index, 0)

total += sum\_

partialsum[i] = total

else: #maxw == bw

do\_cut\_color = 'b'

for i in range(vbox.b1, vbox.b2+1):

sum\_ = 0

for j in range(vbox.r1, vbox.r2+1):

for k in range(vbox.g1, vbox.g2+1):

index = MMCQ.get\_color\_index(j, k, i)

sum\_ += histo.get(index, 0)

total += sum\_

partialsum[i] = total

for i, d in partialsum.items():

lookaheadsum[i] = total - d

#确定切割平面

dim1 = do\_cut\_color + '1'

dim2 = do\_cut\_color + '2'

dim1\_val = getattr(vbox, dim1)

dim2\_val = getattr(vbox, dim2)

for i in range(dim1\_val, dim2\_val+1):

if partialsum[i] > (total / 2):

vbox1 = vbox.copy

vbox2 = vbox.copy

left = i - dim1\_val

right = dim2\_val - i

if left <= right:

d2 = min([dim2\_val - 1, int(i + right / 2)])

else:

d2 = max([dim1\_val, int(i - 1 - left / 2)])

while not partialsum.get(d2, False):

d2 += 1

count2 = lookaheadsum.get(d2)

while not count2 and partialsum.get(d2-1, False):

d2 -= 1

count2 = lookaheadsum.get(d2)

#设置维度

setattr(vbox1, dim2, d2)

setattr(vbox2, dim1, getattr(vbox1, dim2) + 1)

return (vbox1, vbox2)

return (None, None)

@staticmethod

def quantize(pixels, max\_color): #将颜色进行量化

#参数pixels是一个以(r,g,b)形式的像素列表

#参数max\_color是颜色的最大数量

if not pixels:

raise Exception('Empty pixels when quantize.')

if max\_color < 2 or max\_color > 256:

raise Exception('Wrong number of max colors when quantize.')

histo = MMCQ.get\_histo(pixels)

#检查是否低于maxcolors

if len(histo) <= max\_color:

#从histo生成新的颜色并返回

pass

#从颜色重新获取起始vbox

vbox = MMCQ.vbox\_from\_pixels(pixels, histo)

pq = PQueue(lambda x: x.count)

pq.push(vbox)

#实现迭代的内部函数

def iter\_(lh, target):

n\_color = 1

n\_iter = 0

while n\_iter < MMCQ.MAX\_ITERATION:

vbox = lh.pop()

if not vbox.count: #返回

lh.push(vbox)

n\_iter += 1

continue

#实现切割

vbox1, vbox2 = MMCQ.median\_cut\_apply(histo, vbox)

if not vbox1:

raise Exception("vbox1 not defined; shouldn't happen!")

lh.push(vbox1)

if vbox2: #vbox2可以是null

lh.push(vbox2)

n\_color += 1

if n\_color >= target:

return

if n\_iter > MMCQ.MAX\_ITERATION:

return

n\_iter += 1

#第一组颜色，按数量排序

iter\_(pq, MMCQ.FRACT\_BY\_POPULATIONS \* max\_color)

#按像素占用率乘以色彩空间大小的乘积重新排序

pq2 = PQueue(lambda x: x.count \* x.volume)

while pq.size():

pq2.push(pq.pop())

#下一组使用（npix \* vol）排序生成中值切割

iter\_(pq2, max\_color - pq2.size())

#计算实际颜色

cmap = CMap()

while pq2.size():

cmap.push(pq2.pop())

return cmap

class VBox(object):

#3D颜色空间

def \_\_init\_\_(self, r1, r2, g1, g2, b1, b2, histo):

self.r1 = r1

self.r2 = r2

self.g1 = g1

self.g2 = g2

self.b1 = b1

self.b2 = b2

self.histo = histo

@cached\_property

def volume(self):

sub\_r = self.r2 - self.r1

sub\_g = self.g2 - self.g1

sub\_b = self.b2 - self.b1

return (sub\_r + 1) \* (sub\_g + 1) \* (sub\_b + 1)

@property

def copy(self):

return VBox(self.r1, self.r2, self.g1, self.g2,

self.b1, self.b2, self.histo)

@cached\_property

def avg(self):

ntot = 0

mult = 1 << (8 - MMCQ.SIGBITS)

r\_sum = 0

g\_sum = 0

b\_sum = 0

for i in range(self.r1, self.r2 + 1):

for j in range(self.g1, self.g2 + 1):

for k in range(self.b1, self.b2 + 1):

histoindex = MMCQ.get\_color\_index(i, j, k)

hval = self.histo.get(histoindex, 0)

ntot += hval

r\_sum += hval \* (i + 0.5) \* mult

g\_sum += hval \* (j + 0.5) \* mult

b\_sum += hval \* (k + 0.5) \* mult

if ntot:

r\_avg = int(r\_sum / ntot)

g\_avg = int(g\_sum / ntot)

b\_avg = int(b\_sum / ntot)

else:

r\_avg = int(mult \* (self.r1 + self.r2 + 1) / 2)

g\_avg = int(mult \* (self.g1 + self.g2 + 1) / 2)

b\_avg = int(mult \* (self.b1 + self.b2 + 1) / 2)

return r\_avg, g\_avg, b\_avg

def contains(self, pixel):

rval = pixel[0] >> MMCQ.RSHIFT

gval = pixel[1] >> MMCQ.RSHIFT

bval = pixel[2] >> MMCQ.RSHIFT

return all([

rval >= self.r1,

rval <= self.r2,

gval >= self.g1,

gval <= self.g2,

bval >= self.b1,

bval <= self.b2,

])

@cached\_property

def count(self):

npix = 0

for i in range(self.r1, self.r2 + 1):

for j in range(self.g1, self.g2 + 1):

for k in range(self.b1, self.b2 + 1):

index = MMCQ.get\_color\_index(i, j, k)

npix += self.histo.get(index, 0)

return npix

class CMap(object):

#颜色图

def \_\_init\_\_(self):

self.vboxes = PQueue(lambda x: x['vbox'].count \* x['vbox'].volume)

@property

def palette(self):

return self.vboxes.map(lambda x: x['color'])

def push(self, vbox):

self.vboxes.push({

'vbox': vbox,

'color': vbox.avg,

})

def size(self):

return self.vboxes.size()

def nearest(self, color):

d1 = None

p\_color = None

for i in range(self.vboxes.size()):

vbox = self.vboxes.peek(i)

d2 = math.sqrt(

math.pow(color[0] - vbox['color'][0], 2) +

math.pow(color[1] - vbox['color'][1], 2) +

math.pow(color[2] - vbox['color'][2], 2)

)

if d1 is None or d2 < d1:

d1 = d2

p\_color = vbox['color']

return p\_color

def map(self, color):

for i in range(self.vboxes.size()):

vbox = self.vboxes.peek(i)

if vbox['vbox'].contains(color):

return vbox['color']

return self.nearest(color)

class PQueue(object):

#简单优先级队列

def \_\_init\_\_(self, sort\_key):

self.sort\_key = sort\_key

self.contents = []

self.\_sorted = False

def sort(self):

self.contents.sort(key=self.sort\_key)

self.\_sorted = True

def push(self, o):

self.contents.append(o)

self.\_sorted = False

def peek(self, index=None):

if not self.\_sorted:

self.sort()

if index is None:

index = len(self.contents) - 1

return self.contents[index]

def pop(self):

if not self.\_sorted:

self.sort()

return self.contents.pop()

def size(self):

return len(self.contents)

def map(self, f):

return list(map(f, self.contents))

2.音乐部分

FLAGS = tf.app.flags.FLAGS

tf.app.flags.DEFINE\_string('input\_dir', None,

'Directory containing files to convert.') #输入MIDI文件路径

tf.app.flags.DEFINE\_string('output\_file', None,

'Path to output TFRecord file. Will be overwritten '

'if it already exists.') #输出tfrecord文件路径

tf.app.flags.DEFINE\_bool('recursive', False,

'Whether or not to recurse into subdirectories.')

#是否递归查找子路径的文件

tf.app.flags.DEFINE\_string('log', 'INFO',

'The threshold for what messages will be logged '

'DEBUG, INFO, WARN, ERROR, or FATAL.') #显示消息类型

#转换文件

#参数

#root\_dir:指定根目录的字符串

#sub\_dir:一个字符串，指定“根目录”下的路径，

#writer:一个记录

#recursive:一个布尔值，指定是否递归转换文件包含在指定目录的子目录中

#返回:转换文件路径的映射

def convert\_files(root\_dir, sub\_dir, writer, recursive=False):

dir\_to\_convert = os.path.join(root\_dir, sub\_dir)

tf.logging.info("Converting files in '%s'.", dir\_to\_convert)

files\_in\_dir = tf.gfile.ListDirectory(os.path.join(dir\_to\_convert))

recurse\_sub\_dirs = []

written\_count = 0

for file\_in\_dir in files\_in\_dir:

tf.logging.log\_every\_n(tf.logging.INFO, '%d files converted.',

1000, written\_count)

full\_file\_path = os.path.join(dir\_to\_convert, file\_in\_dir)

if (full\_file\_path.lower().endswith('.mid') or

full\_file\_path.lower().endswith('.midi')):

try:

sequence = convert\_midi(root\_dir, sub\_dir, full\_file\_path)

except Exception as exc: #pylint: disable=broad-except

tf.logging.fatal('%r generated an exception: %s',full\_file\_path,exc)

continue

if sequence:

writer.write(sequence)

elif (full\_file\_path.lower().endswith('.xml') or

full\_file\_path.lower().endswith('.mxl')):

try:

sequence = convert\_musicxml(root\_dir, sub\_dir, full\_file\_path)

except Exception as exc: #pylint: disable=broad-except

tf.logging.fatal('%r generated an exception:%s',full\_file\_path, exc)

continue

if sequence:

writer.write(sequence)

elif full\_file\_path.lower().endswith('.abc'):

try:

sequences = convert\_abc(root\_dir, sub\_dir, full\_file\_path)

except Exception as exc: #pylint: disable=broad-except

tf.logging.fatal('%r generated anexception:%s',full\_file\_path,exc)

continue

if sequences:

for sequence in sequences:

writer.write(sequence)

else:

if recursive and tf.gfile.IsDirectory(full\_file\_path):

recurse\_sub\_dirs.append(os.path.join(sub\_dir, file\_in\_dir))

else:

tf.logging.warning(

'Unable to find a converter for file %s', full\_file\_path)

for recurse\_sub\_dir in recurse\_sub\_dirs:

convert\_files(root\_dir, recurse\_sub\_dir, writer, recursive)

#将MIDI文件转换为序列原型

#参数: root\_dir:指定文件根目录的字符串已转换

#sub\_dir：当前正在转换的目录

#full\_file\_path: 要转换文件的完整路径

#return: 如果文件无法转换，则为注释序列原型或无

def convert\_midi(root\_dir, sub\_dir, full\_file\_path):

try:

sequence = midi\_io.midi\_to\_sequence\_proto(

tf.gfile.GFile(full\_file\_path, 'rb').read())

except midi\_io.MIDIConversionError as e:

tf.logging.warning(

'Could not parse MIDI file %s. It will be skipped. Error was: %s',

full\_file\_path, e)

return None #错误处理

sequence.collection\_name = os.path.basename(root\_dir)

sequence.filename = os.path.join(sub\_dir, os.path.basename(full\_file\_path))

sequence.id = note\_sequence\_io.generate\_note\_sequence\_id(

sequence.filename, sequence.collection\_name, 'midi')

tf.logging.info('Converted MIDI file %s.', full\_file\_path)

return sequence

def convert\_directory(root\_dir, output\_file, recursive=False):

#将文件转换为注释序列并写入output\_file

#在根目录中找到的输入文件被转换为带root\_dir的基本名称

#来自root\_dir的文件作为文件名。如果递归为真，递归转换指定目录的任何子目录

#参数:root\_dir指定根目录的字符串

#output\_file:要将结果写入TFRecord文件的路径

#recursive:一个布尔值，指定是否递归转换文件，包含在指定目录的子目录中

with note\_sequence\_io.NoteSequenceRecordWriter(output\_file) as writer:

convert\_files(root\_dir, '', writer, recursive)

#主函数

def main(unused\_argv):

tf.logging.set\_verbosity(FLAGS.log)

#错误处理

if not FLAGS.input\_dir:

tf.logging.fatal('--input\_dir required')

return

if not FLAGS.output\_file:

tf.logging.fatal('--output\_file required')

return

input\_dir = os.path.expanduser(FLAGS.input\_dir) #输入路径

output\_file = os.path.expanduser(FLAGS.output\_file)

#输出文件

output\_dir = os.path.dirname(output\_file) #输出路径

if output\_dir:

tf.gfile.MakeDirs(output\_dir)

convert\_directory(input\_dir, output\_file, FLAGS.recursive)

#运行主函数

def console\_entry\_point():

tf.app.run(main)

将MIDI文件全部存储为tfrecord文件之后，使用polyphony\_rnn\_create\_dataset.py建立数据集，用polyphony模型进行训练，得到音乐数据集。

flags = tf.app.flags

FLAGS = tf.app.flags.FLAGS

flags.DEFINE\_string(

'input', 'E:/college/synaes/midi/midi/tf/pst.tfrecord',

'TFRecord to read NoteSequence protos from.')

#读取NoteSquence的TFReord文件

flags.DEFINE\_string(

'output\_dir', 'E:/college/synaes/poly\_rnn/datasets/pst',

'Directory to write training and eval TFRecord files. The TFRecord files '

'are populated with SequenceExample protos.') #保存序列示例的路径

flags.DEFINE\_float(

'eval\_ratio', 0.1,

'Fraction of input to set aside for eval set. Partition is randomly '

#测试集的比例，划分是随机的

'selected.')

flags.DEFINE\_string(

'log', 'INFO',

'The threshold for what messages will be logged DEBUG, INFO, WARN, ERROR, '

'or FATAL.') #记录调试、信息、警告、错误或致命消息的阈值

#主函数

def main(unused\_argv):

tf.logging.set\_verbosity(FLAGS.log)

pipeline\_instance = polyphony\_rnn\_pipeline.get\_pipeline(

min\_steps=80,

max\_steps=512,

eval\_ratio=FLAGS.eval\_ratio,

config=polyphony\_model.default\_configs['polyphony'])

#配置config为polyphony数据集

input\_dir = os.path.expanduser(FLAGS.input) #输入路径

output\_dir = os.path.expanduser(FLAGS.output\_dir) #输出路径

pipeline.run\_pipeline\_serial(

pipeline\_instance,

pipeline.tf\_record\_iterator(input\_dir, pipeline\_instance.input\_type),

output\_dir) #生成数据集

#运行主函数

def console\_entry\_point():

tf.app.run(main)

7.3.2 模型构建

1.定义模型结构

##### 1)图片情感分析

##### 2)复调音乐模型

class PolyphonyRnnModel(events\_rnn\_model.EventSequenceRnnModel):

#RNN复音序列生成模型类

def generate\_polyphonic\_sequence(

self, num\_steps, primer\_sequence, temperature=1.0, beam\_size=1,

branch\_factor=1, steps\_per\_iteration=1, modify\_events\_callback=None):

#从初级复音轨道生成复音轨道

#参数num\_steps:最后一个轨道的整数长度，以步长为单位，包括引物序列

#primer\_sequence: 引物序列，一个多音序对象

#Temperature: 一个浮点值，指定逻辑值除以多少在计算softmax之前。大于1.0会使轨道更随机，小于1.0则反之

#beam\_size: 一个整数，波束大小在生成轨迹时使用波束搜索

#branch\_factor: 要使用的整数波束搜索分支因子

#steps\_per\_iteration: 一个整数，每次波束搜索需要的步数迭代

#modify\_events\_callback: 用于修改事件列表的可选回调

#返回:生成的复音序列对象

return self.\_generate\_events(num\_steps, primer\_sequence, temperature,

beam\_size, branch\_factor, steps\_per\_iteration,

modify\_events\_callback=modify\_events\_callback)

#返回生成的复音序列对象

def polyphonic\_sequence\_log\_likelihood(self, sequence):

#评估复音序列的对数似然性

#参数sequence：评估日志的复音序列对象的可能性

return self.\_evaluate\_log\_likelihood([sequence])[0]

#返回该模型下序列的对数似然性

#配置模型参数

default\_configs = {

'polyphony': events\_rnn\_model.EventSequenceRnnConfig(

generator\_pb2.GeneratorDetails(

id='polyphony',

description='Polyphonic RNN'), #配置模型为polyphony

magenta.music.OneHotEventSequenceEncoderDecoder(

polyphony\_encoder\_decoder.PolyphonyOneHotEncoding()),

#将复音输入转化成模型之间的输入/输出

contrib\_training.HParams(

batch\_size=64,

rnn\_layer\_sizes=[256, 256, 256],

dropout\_keep\_prob=0.5,

clip\_norm=5,

learning\_rate=0.001)),

#HParams类以名称-值对的形式保存一组超参数

}

7.3.3 模型训练及保存

1.图片情感分析

##### 1)模型训练

def load\_dataset(filename): #加载数据集  
 file\_reader = csv.reader(open(filename, 'rt'), delimiter=',')  
 X, y = [], []  
 for row in file\_reader:  
 X.append(row[0:15]) #获取前15维数据  
 y.append(row[-1]) #获取标签  
 #提取特征名称  
 feature\_names = np.array(X[0])  
 return np.array(X[1:]).astype(np.float32), np.array(y[1:]).astype(np.float32), feature\_names  
if \_\_name\_\_ == '\_\_main\_\_':  
 X,y,feature\_names = load\_dataset('E:/college/synaes/image\_csv/0411.csv')  
 X, y = shuffle(X, y, random\_state=7) #打乱数据  
 num\_training = int(0.9 \* len(X)) #数据的90%作为训练集  
 X\_train, y\_train = X[:num\_training], y[:num\_training]  
 X\_test, y\_test = X[num\_training:], y[num\_training:]  
 rf\_clf = RandomForestClassifier(n\_estimators=1000, max\_depth=10, min\_samples\_split=2) #设置随机森林分类器的参数、决策树的数量、树的深度、最小划分  
 rf\_clf.fit(X\_train, y\_train)  
 y\_pred = rf\_clf.predict(X\_test)  
 print('accuracy:',sklearn.metrics.accuracy\_score(y\_test, y\_pred))  
2)模型保存

joblib.dump(rf\_clf, "E:/college/synaes/image/classifier.m")

模型被保存后，可以被重用，也可以移植到其他环境中使用。

2.音乐训练

##### 1)模型训练

相关代码如下：

FLAGS = tf.app.flags.FLAGS

tf.app.flags.DEFINE\_string('run\_dir', 'E:/college/synaes/poly\_rnn/train\_model/quiet',

'Path to the directory where checkpoints and '

'summary events will be saved during training and '

'evaluation. Separate subdirectories for training '

'events and eval events will be created within '

'`run\_dir`. Multiple runs can be stored within the '

'parent directory of `run\_dir`. Point TensorBoard '

'to the parent directory of `run\_dir` to see all '

'your runs.')

#检查点的保存路径、保存训练和测试过程中的事件，可以通过TensorBoard查看运行状况

tf.app.flags.DEFINE\_string('config', 'polyphony', 'The config to use')

#选择要用的配置

tf.app.flags.DEFINE\_string('sequence\_example\_file', 'E:/college/synaes/poly\_rnn/datasets/quiet'

'/training\_poly\_tracks.tfrecord',

'Path to TFRecord file containing '

#保存有序列示例的TFrecord文件

'tf.SequenceExample records for training or '

'evaluation.')

tf.app.flags.DEFINE\_integer('num\_training\_steps', 0,

'The the number of global training steps your '

#训练步数，0是一直训练直到手动中止

'model should take before exiting training. '

'Leave as 0 to run until terminated manually.')

tf.app.flags.DEFINE\_integer('num\_eval\_examples', 0,

'The number of evaluation examples your model '

'should process for each evaluation step.'

#每次评估用到的训练样本数，0用整个测试样本

'Leave as 0 to use the entire evaluation set.')

tf.app.flags.DEFINE\_integer('summary\_frequency', 10,

'A summary statement will be logged every '

'`summary\_frequency`'

' steps during training or '

'every `summary\_frequency` seconds during '

'evaluation.')

tf.app.flags.DEFINE\_integer('num\_checkpoints', 10,

'The number of most recent checkpoints to keep in '

'the training directory. Keeps all if 0.')

#保存训练目录里最近的检查点数量

tf.app.flags.DEFINE\_boolean('eval', False,

'If True, this process only evaluates the model '

'and does not update weights.')

#如果是True，仅进行测试，不改变模型

tf.app.flags.DEFINE\_string('log', 'INFO',

'The threshold for what messages will be logged '

'DEBUG, INFO, WARN, ERROR, or FATAL.')#容错

tf.app.flags.DEFINE\_string(

'hparams', 'batch\_size=64,rnn\_layer\_sizes=[64,64]',

'Comma-separated list of `name=value` pairs.For each pair, the value of '

'the hyperparameter named `name` is set to `value`. This mapping is merged '

'with the default hyperparameters.')#指定batch的大小和RNN层的大小

#主函数

def main(unused\_argv):

tf.logging.set\_verbosity(FLAGS.log)

#报错提示

if not FLAGS.run\_dir:

tf.logging.fatal('--run\_dir required')

return

if not FLAGS.sequence\_example\_file:

tf.logging.fatal('--sequence\_example\_file required')

return

#打开序列示例

sequence\_example\_file\_paths = tf.gfile.Glob(

os.path.expanduser(FLAGS.sequence\_example\_file))

run\_dir = os.path.expanduser(FLAGS.run\_dir) #保存训练事件

#配置复调音乐模型

config = polyphony\_model.default\_configs[FLAGS.config]

config.hparams.parse(FLAGS.hparams)

mode = 'eval' if FLAGS.eval else 'train'

build\_graph\_fn = events\_rnn\_graph.get\_build\_graph\_fn(

mode, config, sequence\_example\_file\_paths)

#训练模型

train\_dir = os.path.join(run\_dir, 'train')

tf.gfile.MakeDirs(train\_dir)

tf.logging.info('Train dir: %s', train\_dir)

if FLAGS.eval: #是否测试，若为True仅进行测试，不改变模型

eval\_dir = os.path.join(run\_dir, 'eval')

tf.gfile.MakeDirs(eval\_dir)

tf.logging.info('Eval dir: %s', eval\_dir)

num\_batches = (

(FLAGS.num\_eval\_examples or

magenta.common.count\_records(sequence\_example\_file\_paths)) //

config.hparams.batch\_size)

events\_rnn\_train.run\_eval(build\_graph\_fn, train\_dir, eval\_dir, num\_batches)

else: #若为False 则直接训练模型

events\_rnn\_train.run\_training(build\_graph\_fn, train\_dir,

FLAGS.num\_training\_steps,

FLAGS.summary\_frequency,

checkpoints\_to\_keep=FLAGS.num\_checkpoints)

#配置训练模型各项参数（训练步数，保存训练目录里最近的检查点数量，训练样本数等）

#运行主函数

def console\_entry\_point():

tf.app.run(main)

##### 2)模型保存

tf.app.flags.DEFINE\_string('run\_dir', 'E:/college/synaes/poly\_rnn/train\_model/quiet',

'Path to the directory where checkpoints and '

'summary events will be saved during training and '

'evaluation. Separate subdirectories for training '

'events and eval events will be created within '

'`run\_dir`. Multiple runs can be stored within the '

'parent directory of `run\_dir`. Point TensorBoard '

'to the parent directory of `run\_dir` to see all '

'your runs.')

#检查点保存路径、保存训练和测试过程中的事件，可以通过TensorBoard查看运行状况

run\_dir = os.path.expanduser(FLAGS.run\_dir) #保存训练事件

项目8 新闻自动文摘推荐系统

8.3 模块实现

8.3.1 数据预处理

1.导入数据

通过jupyter notebook来实现，相关代码如下：

#导入相应数据包

import pandas as pd

import numpy as np

#数据的读入及读出

df\_news=pd.read\_table("./cnews.val.txt",names=["category","content"])

df\_news.head()#

从文件夹读出相应的数据，分别表示新闻数据的类别及内容，如图8-3所示。

#数据的类别及总量

df\_news.category.unique()

df\_news.content.shape

#为方便后续对数据的处理，将原始表格型据结构转换成列表格式

content\_list=df\_news.content.values.tolist()

2.数据清洗

#jieba分词

content\_fenci = [] #建立一个空的

for line in content\_list:

text = jieba.lcut(line) #给每一条都分词

if len(text) > 1 and text != '\r': #换行

content\_fenci.append(text) #将分词后的结果放入

#content\_fenci[0] #分词后的一个样本

df\_content=pd.DataFrame({'content':content\_fenci})

df\_content.head()

#导入停用词

def drop\_stopwords(contents,stopwords):

content\_clean = [] #放清理后的分词

all\_words = []

for line in contents:

line\_clean=[]

for word in line:

if word in stopwords:

continue

line\_clean.append(word)

all\_words.append(str(word))

content\_clean.append(line\_clean)

return content\_clean,all\_words

content\_clean,all\_words = drop\_stopwords(content\_fenci,stopwords\_list,)

df\_clean= pd.DataFrame({'contents\_clean':content\_clean})

df\_clean.head()

3.统计词频

相关代码如下：

tf= Counter(all\_words)

8.3.2 词云构建

#导入背景图片后的词云

mask = imread('4.png')#读入图片

wc=wordcloud.WordCloud(font\_path=font,mask=mask,background\_color='white',scale=2)

#scale:按照比例进行放大画布，如设置为2，则长和宽都是原来画布的2倍

wc.generate\_from\_frequencies(tf)

plt.imshow(wc) #显示词云

plt.axis('off') #关闭坐标轴

plt.show()

wc.to\_file('ciyun.jpg') #保存词云

8.3.3 关键词提取

import jieba.analyse

index = 2

#print(df\_clean['contents\_clean'][index])

#词之间相连

content\_S\_str = "".join(content\_clean[index])

print(content\_list[index])

print('关键词：')

print(" ".join(jieba.analyse.extract\_tags(content\_S\_str, topK=10, withWeight=False)))

8.3.4 语音播报

将上述提取成功的关键词通过pyttsx 3转换成语音进行播报。

import pyttsx3

voice=pyttsx3.init()

voice.say(" ".join(jieba.analyse.extract\_tags(content\_S\_str, topK=10, withWeight=False)))

print("准备语音播报.....")

voice.runAndWait()

8.3.5 LDA主题模型

def create\_LDA(content\_clean):

#基于文本集建立（词典），并获得特征数

dictionary = corpora.Dictionary(content\_clean)

#基于词典，将分词列表集转换成稀疏向量集，称作语料库

dic = len(dictionary.token2id)

print('词典特征数：%d' % dic)

corpus = [dictionary.doc2bow(sentence) for sentence in content\_clean]

#模型训练

lda = gensim.models.LdaModel(corpus=corpus, id2word = dictionary,num\_topics = 10,passes=10)

#passes 训练几轮

print(lda.print\_topic(1,topn=5))

print('-----------')

for topic in lda.print\_topics(num\_topics=10, num\_words = 5):

print(topic[1])

create\_LDA(content\_clean)

8.3.6 模型构建

df\_train=pd.DataFrame({"content":content\_clean,"label":df\_news['category']})

#为了方便计算，把对应的标签字符类型转换为数字

#映射类型(mapping)

#非空字典

label\_mapping = {"体育": 0, "娱乐": 1, "家居": 2, "房产": 3, "教育":4, "时尚": 5,"时政": 6,"游戏": 7,"科技": 8,"财经": 9}

df\_train['label'] = df\_train['label'].map(label\_mapping)

#df\_train.head()

#将每个新闻信息转换成字符串形式，CountVectorizer和TfidfVectorizer的输入为字符串

def create\_words(data):

words = []

for index in range(len(data)):

try:

words.append( ' '.join(data[index]))

except Exception:

print(index)

return words

#把数据分成测试集和训练集

x\_train,x\_test,y\_train,y\_test =train\_test\_split(df\_train['content'].values,df\_train['label'].values,random\_state=0)

train\_words = create\_words(x\_train)

test\_words = create\_words(x\_test)

#模型训练

#第一种

#CountVectorizer属于常见的特征数值计算类，是一个文本特征提取方法

#对于每个训练文本，只考虑每种词汇在该训练文本中出现的频率

vec = CountVectorizer(analyzer = 'word',max\_features=4000,lowercase=False)

vec.fit(train\_words)

classifier = MultinomialNB()

classifier.fit(vec.transform(train\_words),y\_train)

print("模型准确率:",classifier.score(vec.transform(test\_words), y\_test))

#第二种，TfidfVectorizer除了考量某一词汇在当前训练文本中出现的频率之外

#关注包含这个词汇的其它训练文本数目的倒数，训练文本的数量越多特征化的方法就越有优势

vectorizer = TfidfVectorizer(analyzer='word',max\_features = 40000,

lowercase=False)

vectorizer.fit(train\_words)

classifier.fit(vectorizer.transform(train\_words),y\_train)

print("模型准确率为:",classifier.score(vectorizer.transform(test\_words),

y\_test))

项目9 基于用户特征的预测流量套餐推荐

9.3 逻辑回归算法模块实现

9.3.1 数据预处理

#导入所需库

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import csv

%matplotlib inline

plt.style.use("ggplot")

#导入数据

traindat = pd.read\_csv(open('Train.csv','r'))

testdat = pd.read\_csv(open('Test.csv','r'))

#数据的维度

traindat.shape

#查看数据

traindat.head()

#目标变量在正负样本上的分布是不均匀的

traindat.y.value\_counts()

#每个样本的缺失变量数

traindat.missing\_var = traindat.isnull().sum(axis=1)

testdat.missing\_var = testdat.isnull().sum(axis=1)

#如果部分样本有90%以上的变量是缺失的,直接删除

traindat = traindat.loc[traindat.missing\_var < 100, :]

testdat = testdat.loc[testdat.missing\_var < 100, :]

traindat.shape

#如果某些变量全部为0, 不能提供有效信息,直接删除

traindat = traindat.loc[:, ~(traindat == 0).all(axis=0)]

testdat = testdat.loc[:, ~(testdat == 0).all(axis=0)]

traindat.shape

#非数值型变量

traindat2 = traindat.select\_dtypes(include=['object'])

testdat2 = testdat.select\_dtypes(include=['object'])

traindat2.head(3)

数据集中的非数值型变量

traindat2.shape

type(traindat2)

pandas.core.frame.DataFrame

#仅抽取数值型变量

traindat1 = traindat.select\_dtypes(exclude=['object'])

testdat1 = testdat.select\_dtypes(exclude=['object'])

traindat1.head(3)

数据集中的非数值型变量

#用均值填补缺失的数值变量

traindat1 = traindat1.fillna(traindat1.mean())

testdat1 = testdat1.fillna(testdat1.mean())

traindat1.shape

from sklearn.preprocessing import LabelEncoder

var\_mod1 = ['job','marital','education','default','housing','loan','contact','month','day\_of\_week','poutcome','y']

var\_mod11 = ['job','marital','education','default','housing','loan','contact','month','day\_of\_week','poutcome']

le1 = LabelEncoder()

for i in var\_mod1:

traindat2[i] = le1.fit\_transform(traindat2[i])

#对训练集先拟合，然后转换，实现数据的标准化、归一化

for i in var\_mod11:

testdat2[i] = le1.fit\_transform(testdat2[i])

#对测试集先拟合,然后转换，实现数据的标准化，归一化

traindat2.dtypes

traindat2.head()

testdat2.head()

将训练集中的非数值型变量转化为数值型变量

traindat1 = traindat1.join(traindat2)

#非数值变量与数值变量一起构成原来的训练集

testdat1 = testdat1.join(testdat2)

#非数值变量与数值变量一起构成原来的测试集

#定义X与y方便建模

x, y = traindat1.iloc[:, 0:-1], traindat1.y #去最后两列数据

xtest = testdat1.iloc[:, :] #去所有行，所有列

x.shape, y.shape, xtest.shape

x.head() #查看训练集中的特征

xtest.head()#查看验证集的特征

由于版面限制，只选取部分列进行展示

#建模所需库

from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier

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

from sklearn.model\_selection import GridSearchCV

from sklearn.metrics import confusion\_matrix, classification\_report, roc\_curve, auc

#正负样本比例

np.mean(y == 0), np.mean(y == 1)

#训练集和验证集拆分

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x.values, y.values, test\_size=0.3)

x\_train.shape, y\_train.shape, x\_test.shape, y\_test.shape

#训练集中的正负样本比例

np.mean(y\_train == 0), np.mean(y\_train == 1)

9.3.2 模型构建

1.构建模型

相关代码如下：

#构建一个逻辑回归模型, 采用默认参数设置

lr\_clf1 = LogisticRegression(class\_weight={0: 0.11, 1: 0.89})

#以0，0.11，1，0.89类型权重参数去构建模型

lr\_clf1.fit(x\_train, y\_train) #拟合训练x,y

y\_train\_pred = lr\_clf1.predict(x\_train) #预测y值

print("Confusion matrix (training):\n {0}\n".format(confusion\_matrix(y\_train, y\_train\_pred)))

#输出混淆矩阵结果

print("Classification report (training):\n {0}".format(classification\_report(y\_train, y\_train\_pred)))

#输出分类报告结果

y\_test\_pred = lr\_clf1.predict(x\_test)

print("Confusion matrix (validation):\n {0}\n".format(confusion\_matrix(y\_test, y\_test\_pred)))

#测试输出混淆矩阵结果

print("Classification report (validation):\n {0}".format(classification\_report(y\_test, y\_test\_pred)))

#测试输出分类报告结果

2.优化模型

相关代码如下：

#参数调整

lr\_clf\_tuned = LogisticRegression(class\_weight={0: 0.03, 1: 0.97})

lr\_clf\_params = {

"penalty": ["l1", "l2"],

"C": [1, 1.3, 1.5, 1.7, 2]

}

lr\_clf\_cv = GridSearchCV(lr\_clf\_tuned, lr\_clf\_params, cv=5)

lr\_clf\_cv.fit(x\_train, y\_train)

print(lr\_clf\_cv.best\_params\_)

9.3.3 模型评估及保存

#采用最优参数构建逻辑回归模型

lr\_clf2 = LogisticRegression(penalty="l2", C=1, class\_weight={0: 0.11, 1: 0.89}) #增加正则化项,正则化强度1

lr\_clf2.fit(x\_train, y\_train) #拟合x,y

y\_train\_pred = lr\_clf2.predict(x\_train)

print("Confusion matrix (training):\n {0}\n".format(confusion\_matrix(y\_train, y\_train\_pred)))

#输出混淆矩阵结果

print("Classification report (training):\n {0}".format(classification\_report(y\_train, y\_train\_pred)))

#输出分类报告结果

y\_test\_pred = lr\_clf2.predict(x\_test)

print("Confusion matrix (validation):\n {0}\n".format(confusion\_matrix(y\_test, y\_test\_pred)))

print("Classification report (validation):\n {0}".format(classification\_report(y\_test, y\_test\_pred)))

#绘制ROC曲线

y1\_valid\_score\_lr2 = lr\_clf2.predict\_proba(x\_test)

fpr\_lr2, tpr\_lr2, thresholds\_lr2 = roc\_curve(y\_test, y1\_valid\_score\_lr2[:, 1])

roc\_auc\_lr2 = auc(fpr\_lr2, tpr\_lr2)

plt.plot(fpr\_lr2, tpr\_lr2, lw=2, alpha=.6)

plt.plot([0, 1], [0, 1], lw=2, linestyle="--")

plt.xlim([0, 1])

plt.ylim([0, 1.05])

plt.xlabel("误报率")

plt.ylabel("命中率")

plt.title("逻辑回归算法的ROC曲线")

plt.legend(["(AUC {:.4f})".format(roc\_auc\_lr2)], fontsize=9, loc=2)

<matplotlib.legend.Legend at 0x1fd985ed0f0>

#绘制逻辑回归算法的ROC曲线

9.3.4 模型预测

相关代码如下：

#对test的结果进行输出，并保存为Results.csv文件

ytest\_pred = lr\_clf2.predict(xtest)

type(ytest\_pred)

print(ytest\_pred)

from pandas import Series, DataFrame

predictY = DataFrame(ytest\_pred,columns=['y'])

predictY.to\_csv('Results.csv', encoding = 'utf-8', index=False , header=False) #对预测结果输出为Results表格形式

9.4朴素贝叶斯算法模型实现

9.4.1 数据预处理

相关代码如下：

#导入所需库

from collections import Counter

import numpy as np

import pandas as pd

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

#用于随机划分训练子集和测试子集

import matplotlib.pyplot as plt #导入绘图库

import math

from sklearn.metrics import roc\_auc\_score #用于绘制ROC曲线并计算AUC值

from sklearn.metrics import confusion\_matrix #混淆矩阵

from sklearn.metrics import classification\_report

#返回精确度、召回率及F1值

from sklearn import metrics #便于调用各种评价指标函数

%matplotlib inline

plt.style.use("ggplot")

#画图观测数学型数据分布

def plot(counter):

d = dict(counter)

plt.bar(d.keys(),d.values())

plt.show()

data = pd.read\_csv("Train.csv",encoding = "utf-8")

data = data.drop('duration',axis=1)

#观测数据分布过程

print(Counter(data[data['y']=='no']['nr.employed']))

#标签为“no”的数据中特征“nr.employed”的分布

print(Counter(data[data['y']=='yes']['nr.employed']))

#标签为“yes”的数据中特征“nr.employed”的分布

plot(Counter(data[data['y']=='no']['nr.employed'])) #绘制图像

plot(Counter(data[data['y']=='yes']['nr.employed'])) #部分数据特征分布

#对特征pdays进行二分类

data.loc[data['pdays']!=999,"pdays"] = 1

data.loc[data['pdays']==999,"pdays"] = 0

#对euribor3m（同行拆借利率）特征进行区域划分

range\_eu=sorted(dict(Counter(data[data['y']=='yes']['euribor3m'])).keys())

eu = 0.5

#将连续型的分布按照区域划分，转化连续为离散

for i in range(10):

data.loc[(range\_eu[0]+eu\*i <=data['euribor3m']) & (data['euribor3m']<range\_eu[0]+eu\*(i+1)),'euribor3m'] = range\_eu[0]+eu\*i+eu\*1/2

#绘制分布直方图

plot(Counter(data[data['y']=='no']['euribor3m']))

plot(Counter(data[data['y']=='yes']['euribor3m']))

#将连续型分布转化为离散型分布

#对特征cons.conf.idx区间划分，化连续为离散

range\_conf = sorted(dict(Counter(data[data['y']=='yes']['cons.conf.idx'])).keys())

para = 1

for i in range(15):

data.loc[(range\_conf[0]+para\*i <=data['cons.conf.idx']) & (data['cons.conf.idx']<range\_conf[0]+para\*(i+1)),'cons.conf.idx'] = range\_conf[0]+para\*i+para\*1/2

#去除某些连续值特征（正态分布、几何分布），只保存离散值特征

tra\_feature = list(data.columns)

tra\_feature.remove('cons.price.idx')

tra\_feature.remove('campaign')

print(tra\_feature)

#已知标签的数据按4:1拆分成训练集和验证集

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data[data.columns[:-1]], data[data.columns[-1]], test\_size=0.2, random\_state=0)

#训练集中的yes和no样本比例

y\_pro ={}

d=dict(Counter(y\_train))

for key in d:

d[key] = d[key]/len(X\_train)

y\_pro = d

print(y\_pro)

#yes条件概率列表（内部元素为各特征的字典）

feature\_yes\_pro = []

X\_train\_yes = X\_train[y\_train=='yes']

for x in tra\_feature[:-1]:

d = dict(Counter(X\_train\_yes[x])) #计算某一特征中各取值的数量

for key in d:

d[key] = d[key]/len(X\_train\_yes) #计算某一特征中各取值的条件概率

feature\_yes\_pro.append(d)

print("yes条件概率列表")

print(feature\_yes\_pro[1]) #打印标签为yes的数据中特征1的条件概率分布

print('\n')

print(feature\_yes\_pro[2])

print('\n')

print(feature\_yes\_pro[3])

print('\n')

print("......")

print('\n')

#no条件概率列表（内部元素为各特征的字典）

feature\_no\_pro = []

X\_train\_no = X\_train[y\_train=='no']

for x in tra\_feature[:-1]:

d = dict(Counter(X\_train\_no[x]))

for key in d:

d[key] = d[key]/len(X\_train\_no)

feature\_no\_pro.append(d)

print(feature\_no\_pro[1])

print('\n')

print(feature\_no\_pro[2])

print('\n')

print(feature\_no\_pro[3])

print('\n')

print("......")

9.4.2 模型构建

相关代码如下：

#朴素贝叶斯模型建立

X\_test\_l = np.array(X\_test.drop(['cons.price.idx','campaign'],axis = 1,inplace=False)).tolist()

result = []

result\_bool=[]

index = 0

#将验证集中的数据逐条计算后验概率

for i in X\_test\_l:

num = 0

p\_yes = 1

p\_no =1

#朴素贝叶斯公式的实现

for x in i:

if not (x in feature\_yes\_pro[num].keys()):

a\_yes\_pro = 1/(len(X\_train\_yes)+2)

if not (x in feature\_no\_pro[num].keys()):

a\_no\_pro = 1/(len(X\_train\_no)+2)

else: a\_no\_pro=(dict(Counter(X\_train\_no[tra\_feature[num]]))[x]+1)/(len(X\_train\_no)+2)

p\_yes = p\_yes\*a\_yes\_pro

p\_no = p\_no\*a\_no\_pro

num = num+1

continue

if not (x in feature\_no\_pro[num].keys()):

a\_no\_pro = 1/(len(X\_train\_no)+2)

if not (x in feature\_yes\_pro[num].keys()):

a\_yes\_pro = 1/(len(X\_train\_yes)+2)

else: a\_yes\_pro=(dict(Counter(X\_train\_yes[tra\_feature[num]]))[x]+1)/(len(X\_train\_yes)+2)

p\_yes = p\_yes\*a\_yes\_pro

p\_no = p\_no\*a\_no\_pro

num = num+1

continue

p\_yes = p\_yes\*feature\_yes\_pro[num][x]

p\_no = p\_no\*feature\_no\_pro[num][x]

num = num+1

#结果概率

p\_yes = p\_yes\*y\_pro['yes']

p\_no = p\_no\*y\_pro['no']

index = index+1

if p\_yes>=p\_no: #取后验概率中较大值所对应的标签作为估计结果

result\_bool.append(1)

else:

result\_bool.append(0)

result.append(p\_yes/(p\_yes+p\_no))

9.4.3 模型评估及保存

相关代码如下：

#对模型拟合程度进行检验

y\_test\_bool = list(y\_test)

for i in range(len(y\_test\_bool)):

if y\_test\_bool[i]=='yes': #将预测结果与标签进行比对

y\_test\_bool[i]=1

else:

y\_test\_bool[i]=0

y\_test\_bool = np.array(y\_test\_bool)

result = np.array(result)

print(roc\_auc\_score(y\_test\_bool, result)) #计算并打印AUC值

print(confusion\_matrix(y\_test\_bool, result\_bool, labels=None, sample\_weight=None)) #打印混淆矩阵

print(classification\_report(y\_test\_bool, result\_bool))

#打印分类指标文本报告

#绘制ROC曲线

fpr, tpr, threshold = metrics.roc\_curve(y\_test\_bool, result)

roc\_auc = metrics.auc(fpr, tpr)

plt.plot(fpr, tpr, lw=2, alpha=.6)

plt.plot([0, 1], [0, 1], lw=2, linestyle="--")

plt.xlim([0, 1])

plt.ylim([0, 1])

plt.xlabel("误报率")

plt.ylabel("命中率")

plt.title("ROC 曲线")

plt.legend(["朴素贝叶斯算法 (AUC {:.4f})".format(roc\_auc)],fontsize=10, loc=2)

<matplotlib.legend.Legend at 0x2f6fae3b808>

#绘制朴素贝叶斯算法的ROC曲线

项目10 校园知识图谱问答推荐系统

10.3 模块实现

10.3.1 构造数据集

加载训练集相关代码如下：

def \_read\_data(cls, input\_file):

#读取数据集文件

with codecs.open(input\_file,'r',encoding='utf-8') as f:

lines = []

words = []

labels = []

for line in f:

contends = line.strip()

tokens = contends.split('\t')

if len(tokens) == 2:

words.append(tokens[0])

labels.append(tokens[1])

else:

if len(contends) == 0:

l=''.join([label for label in labels if len(label) > 0])

w = ' '.join([word for word in words if len(word) > 0])

lines.append([l, w])

words = []

labels = []

continue

if contends.startswith("-DOCSTART-"):

words.append('')

continue

return lines

#读取训练集

def get\_train\_examples(self, data\_dir):

return self.\_create\_example(

self.\_read\_data(os.path.join(data\_dir, "train.txt")), "train"

)

#读取验证集

def get\_dev\_examples(self, data\_dir):

return self.\_create\_example(

self.\_read\_data(os.path.join(data\_dir,"dev.txt")),"dev"

)

#读取测试集

def get\_test\_examples(self, data\_dir):

return self.\_create\_example(

self.\_read\_data(os.path.join(data\_dir, "test.txt")), "test")

10.3.2 识别网络

def train\_ner(): #定义训练

import os

from bert\_base.train.train\_helper import get\_args\_parser

from bert\_base.train.bert\_lstm\_ner import train

args = get\_args\_parser()

if True:

import sys

param\_str = '\n'.join(['%20s = %s' % (k, v) for k, v in sorted(vars(args).items())])

print('usage: %s\n%20s %s\n%s\n%s\n' % (' '.join(sys.argv), 'ARG', 'VALUE', '\_' \* 50, param\_str))

print(args)

os.environ['CUDA\_VISIBLE\_DEVICES'] = args.device\_map

train(args=args)

#数据处理代码

def convert\_single\_example(ex\_index, example, label\_list, max\_seq\_length, tokenizer, output\_dir, mode):

#将一个样本进行分析，字和标签转化为ID，结构化到输入特征对象中

label\_map = {}

#1表示从1开始对标签进行索引化

for (i, label) in enumerate(label\_list, 1):

label\_map[label] = i

#保存label->index 的映射

if not os.path.exists(os.path.join(output\_dir, 'label2id.pkl')):

with codecs.open(os.path.join(output\_dir,'label2id.pkl'),'wb')as w:

pickle.dump(label\_map, w)

textlist = example.text.split(' ')

labellist = example.label.split(' ')

tokens = []

labels = []

for i, word in enumerate(textlist):

#分词，不在BERT的vocab.txt中，则进行WordPiece处理，分字可替换为list(input)

token = tokenizer.tokenize(word)

tokens.extend(token)

label\_1 = labellist[i]

for m in range(len(token)):

if m == 0:

labels.append(label\_1)

else: #一般不会出现else分支

labels.append("X")

#tokens = tokenizer.tokenize(example.text)

#序列截断

if len(tokens) >= max\_seq\_length - 1:

tokens = tokens[0:(max\_seq\_length - 2)]

#-2的原因是因为序列需要加一个句首和句尾标志

labels = labels[0:(max\_seq\_length - 2)]

ntokens = []

segment\_ids = []

label\_ids = []

ntokens.append("[CLS]") #句子开始设置CLS标志

segment\_ids.append(0)

#append("O") or append("[CLS]") not sure!

label\_ids.append(label\_map["[CLS]"])

#O或者CLS会减少标签个数，但句首和句尾使用不同的标志标注

for i, token in enumerate(tokens):

ntokens.append(token)

segment\_ids.append(0)

label\_ids.append(label\_map[labels[i]])

ntokens.append("[SEP]") #句尾添加[SEP]标志

segment\_ids.append(0)

#append("O") or append("[SEP]") not sure!

label\_ids.append(label\_map["[SEP]"])

input\_ids = tokenizer.convert\_tokens\_to\_ids(ntokens)

#将序列中的字(ntokens)转化为ID形式

input\_mask = [1] \* len(input\_ids)

#label\_mask = [1] \* len(input\_ids)

#使用padding

while len(input\_ids) < max\_seq\_length:

input\_ids.append(0)

input\_mask.append(0)

segment\_ids.append(0)

label\_ids.append(0)

ntokens.append("\*\*NULL\*\*")

#label\_mask.append(0)

#print(len(input\_ids))

assert len(input\_ids) == max\_seq\_length

assert len(input\_mask) == max\_seq\_length

assert len(segment\_ids) == max\_seq\_length

assert len(label\_ids) == max\_seq\_length

#assert len(label\_mask) == max\_seq\_length

#打印部分样本数据信息

if ex\_index < 5:

tf.logging.info("\*\*\* Example \*\*\*")

tf.logging.info("guid: %s" % (example.guid))

tf.logging.info("tokens: %s" % " ".join(

[tokenization.printable\_text(x) for x in tokens]))

tf.logging.info("input\_ids:%s"% " ".join([str(x) for x in input\_ids]))

tf.logging.info("input\_mask: %s" % " ".join([str(x) for x in input\_mask]))

tf.logging.info("segment\_ids: %s" % " ".join([str(x) for x in segment\_ids]))

tf.logging.info("label\_ids: %s" % " ".join([str(x) for x in label\_ids]))

# tf.logging.info("label\_mask: %s" % " ".join([str(x) for x in label\_mask]))

#结构化为一个类

feature = InputFeatures(

input\_ids=input\_ids,

input\_mask=input\_mask,

segment\_ids=segment\_ids,

label\_ids=label\_ids,

#label\_mask = label\_mask

)

#mode='test'的时候才有效

write\_tokens(ntokens, output\_dir, mode)

return feature

10.3.3 命名实体纠错

class Select\_course:

def \_\_init\_\_(self):

self.f = csv.reader(open('QA/dict/course.txt','r'))

self.course\_name = [i[0].strip() for i in self.f]

self.led = 3

self.limit\_num = 10

self.select\_word = []

self.is\_same = False

self.have\_same\_length = False

self.input\_word = ''

self.is\_include = False

#print(self.course\_name)

#print('列表创建完毕....')

#包含搜索

def select\_first(self, input\_word):

self.select\_word = []

self.is\_same = False

self.is\_include = False

self.have\_same\_length = False

self.input\_word = input\_word

if input\_word in self.course\_name:

self.is\_same = True

self.select\_word.append(input\_word)

if self.is\_same == False:

for i in self.course\_name:

mark = True

for one\_word in input\_word:

if not one\_word in i:

mark = False

if mark:

self.select\_word.append(i)

if len(self.select\_word) != 0:

self.is\_include = True

#print('第一轮筛选:')

#print(self.select\_word)

#模糊搜索

def select\_second(self):

self.led = 3

if self.is\_same or self.is\_include:

return

for name in self.course\_name:

ed = ls.distance(self.input\_word, name)

if ed <= self.led:

self.led = ed

self.select\_word.append(name)

select\_word\_copy1 = copy.deepcopy(self.select\_word)

for name in select\_word\_copy1:

ed = ls.distance(self.input\_word, name)

if ed > self.led:

self.select\_word.remove(name)

if ed == self.led and len(name) == len(self.input\_word):

self.hava\_same\_length = True

#print('第二轮筛选:')

#print(self.select\_word)

class Select\_name:

def \_\_init\_\_(self): #定义初始化

self.f = csv.reader(open('QA/dict/teacher.csv','r'))

self.teacher\_name = [i[0] for i in self.f]

self.led = 3

self.limit\_num = 10

self.select\_word = []

self.have\_same\_length = False

self.is\_same = False

self.input\_word = ''

#print(self.teacher\_name)

#print('列表创建完毕....')

def select\_first(self, input\_word): #定义首选

self.select\_word = []

self.have\_same\_length = False

self.is\_same = False

self.input\_word = input\_word

if input\_word in self.teacher\_name:

self.is\_same = True

self.select\_word.append(input\_word)

if self.is\_same == False:

for name in self.teacher\_name:

ed = ls.distance(self.input\_word, name)

if ed <= self.led:

self.led = ed

self.select\_word.append(name)

select\_word\_copy1 = copy.deepcopy(self.select\_word)

for name in select\_word\_copy1:

ed = ls.distance(self.input\_word, name)

if ed > self.led:

self.select\_word.remove(name)

if ed == self.led and len(name) == len(self.input\_word):

self.hava\_same\_length = True

#print('第一轮筛选:')

#print(self.select\_word)

return

def select\_second3(self): #定义后续筛选

if self.is\_same == True or len(self.input\_word) != 3:

return

select\_word\_copy2 = copy.deepcopy(self.select\_word)

if self.hava\_same\_length:

for name in select\_word\_copy2:

if len(self.input\_word)!=len(name):

self.select\_word.remove(name)

#print('第二轮筛选:')

#print(self.select\_word)

def select\_third3(self):

if self.is\_same == True or len(self.input\_word) != 3:

return

select\_word\_copy3 = copy.deepcopy(self.select\_word)

self.select\_word = []

for name in select\_word\_copy3:

if name[0] == self.input\_word[0] and name[2] == self.input\_word[2]:

self.select\_word.append(name)

for name in select\_word\_copy3:

if not(name[0]==self.input\_word[0]and name[2]== self.input\_word[2]):

self.select\_word.append(name)

#print('第三轮筛选:')

#print(self.select\_word)

def limit\_name\_num(self):

while(len(self.select\_word)>self.limit\_num):

self.select\_word.pop()

#print('列表大小限制:')

#print(self.select\_word)

10.3.4 检索问题类别

相关代码如下：

if self.check\_words(self.direction\_qwds,question)and('teacher' in types):     question\_type = 'teacher\_direction'

     question\_types.append(question\_type)

  if self.check\_words(self.location\_qwds, question)and ('teacher' in types):     question\_type = 'teacher\_location'

     question\_types.append(question\_type)

  if self.check\_words(self.telephone\_qwds,question)and ('teacher' in types):   question\_type = 'teacher\_telephone'

     question\_types.append(question\_type)

10.3.5 查询结果

相关代码如下：

if final\_question\_type == 'teacher\_direction':

    sql = "MATCH (m:Teacher) where m.name = '{0}' return m.name, m.research\_direction".format(i)

  if final\_question\_type == 'teacher\_location':

    sql = "MATCH (m:Teacher) where m.name = '{0}' return m.name, m.office\_location".format(i)

  if final\_question\_type == 'teacher\_telephone':

    sql = "MATCH (m:Teacher) where m.name = '{0}' return m.name, m.telephone".format(i)

#连接数据库

def \_\_init\_\_(self):

   self.g = Graph(

         "http://10.3.55.50:7474/browser",

         user="\*\*\*\*\*\*\*\*",

         password="\*\*\*\*\*\*\*\*")

      self.num\_limit = 30

#查询结果并返回编写的模版答案语句

def search\_main(self, sqls, final\_question\_types):

      final\_answers = []

      temp\_data = []

      data = []

      for i in sqls:

          for one\_sql in i:

              temp\_data.append(self.g.run(one\_sql).data()[0])

              #print(temp\_data)

          data.append(temp\_data)

          temp\_data = []

      #print(data)

      temp\_answer = []

      answer = []

      for i in zip(final\_question\_types, data):

          for one\_type\_and\_data in zip(i[0],i[1]):

         temp\_answer.append(self.answer\_prettify(one\_type\_and\_data[0],one\_type\_and\_data[1]))

          answer.append(temp\_answer)

          temp\_answer = []

      return answer

重复询问以剔除错误的备选

ask\_again = ''

final\_question\_types = []

for i in zip(tags, pre\_words):

     #print(i)

     if len(i[1]) == 1:

         final\_question\_types.append(classifier.classify(text, i[0]))

         final\_words.append(i[1][0])

      if len(i[1]) > 1:

         print('>1')

         if i[0] == 'teacher':

             ask\_again = '请问您要询问的是哪个老师的信息：{0}'.format(','.join(i[1]))

         if i[0] ==  'course':

             ask\_again = '请问您要询问的是哪门课程的信息：{0}'.format(','.join(i[1]))

         #print(ask\_again)

         answer\_again = input(ask\_again)

         final\_words.append(answer\_again)

         final\_question\_types.append(classifier.classify(text, i[0]))

项目11 新闻推荐系统

11.3 模块实现

11.3.1 数据预处理

11.3.2 热度值计算

新闻热度值计算对应的函数为：

def calHotValue(self):

base\_time = datetime.now()

sql = "select new\_id, new\_cate\_id, new\_seenum, new\_disnum, new\_time from new"

self.cursor.execute(sql)

result\_list = self.cursor.fetchall()

result = list()

for row in result\_list:

diff=base\_time-datetime.strptime(str(row[4].date()),'%Y-%m-%d')

hot\_value = row[2] \* 0.4 + row[3] \* 0.5 - diff.days \* 0.1

result.append((row[0],row[1],hot\_value))

return result

11.3.3 相似度计算

1.新闻分词处理

使用Python的xlrd.open\_work()函数加载Excel文件。加载原始数据对应的函数实现为：

#加载数据

def loadData(self):

news\_dict = dict()

#使用xlrd加载xlsx格式文件,返回一个table对象

table = xlrd.open\_workbook(self.file).sheets()[0]

#遍历每一行

for row in range(1,table.nrows):

#将每一列返回为一个数组

line = table.row\_values(row, start\_colx=0, end\_colx=None)

new\_id = int(line[0])

news\_dict.setdefault(new\_id,{})

news\_dict[new\_id]["tag"]= line[1]

news\_dict[new\_id]["title"] = line[5]

news\_dict[new\_id]["content"] = line[-1]

return news\_dict

原始数据加载之后保存在变量news\_dict中，在文章标题分词时使用，分词使用的是jieba.analyse.extract\_tags()函数。

#调用jieba分词获取每篇文章的关键词

def getKeyWords(self):

news\_key\_words = list()

#加载停用词表

stop\_words\_list=[line.strip()for line in open

("./../files/stop\_words.txt").readlines()]

for new\_id in self.news\_dict.keys():

if self.\_type == 1:

#allowPOS 提取地名、名词、动名词、动词

keywords = jieba.analyse.extract\_tags(

self.news\_dict[new\_id]["title"]

+self.news\_dict[new\_id]["content"],

topK=10,

withWeight=False,

allowPOS=('ns', 'n', 'vn', 'v')

)

news\_key\_words.append(str(new\_id)+'\t'+",".join(keywords))

elif self.\_type == 2:

#cut\_all :False 表示精确模式

keywords=jieba.cut(

self.news\_dict[new\_id]["title"],cut\_all=False)

kws = list()

for kw in keywords:

if kw not in stop\_words\_list and kw != " "

and kw != " ":

kws.append(kw)

news\_key\_words.append(str(new\_id)+'\t'+

",".join(kws))

else:

print("请指定获取关键词的方法类型<1：TF-IDF 2：标题分词法>")

return news\_key\_words

2.计算相似度

新闻相似度的计算采用杰卡德相似系数，其对应函数为：

def getCorrelation(self):

news\_cor\_list = list()

for newid1 in self.news\_tags.keys():

id1\_tags = set(self.news\_tags[newid1].split(","))

for newid2 in self.news\_tags.keys():

id2\_tags = set(self.news\_tags[newid2].split(","))

if newid1 != newid2:

print( newid1 + "\t" + newid2 + "\t" +

str(id1\_tags & id2\_tags) )

cor = ( len(id1\_tags & id2\_tags) ) / len

(id1\_tags | id2\_tags)

if cor > 0.0:

news\_cor\_list.append([newid1,newid2,

format(cor,".2f")])

return news\_cor\_list

11.3.4 新闻统计

相关代码如下：

#获取每个标签下对应的文章

def getNewsTags(self):

result = dict()

for file in os.listdir(self.kw\_path):

path = self.kw\_path + file

for line in open(path, encoding= "utf-8").readlines():

try:

newid, tags = line.strip().split("\t")

except:

print("%s 下无对应标签" % newid)

for tag in tags.split(","):

if tag in ALLOW\_TAGS:

sql = "select new\_hot from newhot where new\_id=%s" % newid

self.cursor.execute(sql)

hot\_value = self.cursor.fetchone()

result.setdefault(tag,{})

result[tag][newid]=hot\_value[0]

return result

#对每个标签下的新闻进行排序，并写入mysql

def writeToMySQL(self):

for tag in self.result.keys():

for newid in self.result[tag].keys():

sql\_w = "insert into newtag( new\_tag,new\_id,new\_hot ) values('%s', '%s' ,%s)"

% (tag, newid, self.result[tag][newid])

try:

self.cursor.execute(sql\_w)

self.db.commit()

except:

print("rollback", tag,newid,self.result[tag][newid])

self.db.rollback()

11.3.5 API接口开发

相关代码如下：

def home(request):

#从前端请求中获取cateid

\_cate = request.GET.get("cateid")

if "username" not in request.session.keys():

return JsonResponse({ "code":0 })

total = 0 #总页数

#如果cate 是为你推荐，走该部分逻辑tag\_flag = 0表示不是从标签召回数据

if \_cate == "1":

news, news\_hot\_value = getRecNews(request)

#如果cate 是热度榜，走该部分逻辑

elif \_cate == "2":

news,news\_hot\_value = getHotNews()

#其他正常的请求获取

else:

\_page\_id = int(request.GET.get("pageid"))

news = new.objects.filter(new\_cate=\_cate).order\_by("-new\_time")

total = news.\_\_len\_\_()

news = news[\_page\_id \* 10:(\_page\_id+1) \* 10]

#数据拼接

result = dict()

result["code"] = 2

result["total"] = total

result["cate\_id"] = \_cate

result["cate\_name"] = str(cate.objects.get(cate\_id=\_cate))

result["news"] = list()

for one in news:

result["news"].append({

"new\_id":one.new\_id,

"new\_title":str(one.new\_title),

"new\_time": one.new\_time,

"new\_cate": one.new\_cate.cate\_name,

"new\_hot\_value": news\_hot\_value[one.new\_id] if \_cate == "2" or \_cate == "1" else 0 ,

"new\_content": str(one.new\_content[:100])

})

return JsonResponse(result)

相关函数如下：

#热度榜排序逻辑：new\_seenum\*0.3+new\_disnum\*0.5+

(new\_date-base\_data)\* 0.2

def getHotNews():

#从新闻热度表中取top 20数据

all\_news=newhot.objects.order\_by("new\_hot").

values("new\_id","new\_hot")[:20]

all\_news\_id = [one["new\_id"] for one in all\_news]

all\_news\_hot\_value = { one["new\_id"]:one["new\_hot"]

for one in all\_news}

#返回热度榜单数据

return new.objects.filter(new\_id\_\_in=all\_news\_id),

all\_news\_hot\_value

#为你推荐的数据获取逻辑

def getRecNews(request):

tags = request.GET.get('tags')

baseclick = request.GET.get("baseclick")

tag\_flag = 0 if tags == "" else 1

tags\_list= tags.split(",")

uname = request.session["username"]

#标签召回逻辑

if tag\_flag == 1 and int(baseclick) == 0:

num = (20 / len(tags\_list)) + 1

news\_id\_list = list()

news\_id\_hot\_dict = dict()

for tag in tags\_list:

result= newtag.objects.filter

(new\_tag=tag).values("new\_id","new\_hot")[:num]

for one in result:

news\_id\_list.append(one["new\_id"])

news\_id\_hot\_dict[one["new\_id"]] = one["new\_hot"]

return new.objects.filter(new\_id\_\_in=news\_id\_list)

[:20], news\_id\_hot\_dict

#正常排序逻辑

elif tag\_flag ==0:

#首先判断用户是否有浏览记录

#如果有该用户的浏览记录，则从浏览的新闻获取相似的新闻返回

if newbrowse.objects.filter(user\_name=uname).exists():

#判断用户浏览的新闻是否够10个，如果够每个取两个相似，不够则每个取20/真实个数+1相似

num = 0

browse\_dict = newbrowse.objects.filter

(user\_name=uname).order\_by

("new\_browse\_time").values("new\_id")[:10]

if browse\_dict.\_\_len\_\_() < 10:

num = ( 20 / browse\_dict.\_\_len\_\_()) +1

else:

num = 2

news\_id\_list = list()

all\_news\_hot\_value = dict()

#遍历最近浏览的N篇新闻，每篇新闻取num篇相似新闻

for browse\_one in browse\_dict:

for one in newsim.objects.filter

(new\_id\_base=browse\_one["new\_id"]).order\_by("-new\_correlation")

[:num]:news\_id\_list.append(one.new\_id\_sim)all\_news\_hot\_value

[one.new\_id\_sim]= (newhot.objects.filter(new\_id=browse\_one

["new\_id"])[0]).new\_hot

return new.objects.filter(new\_id\_\_in=news\_id\_list)[:20], all\_news\_hot\_value

#如果该用户没有浏览记录，第一次进入系统且没有选择任何标签，返回热度榜单数据的20～40

else:

#从新闻热度表中取top20 新闻数据

all\_news = newhot.objects.order\_by("-new\_hot").values

("new\_id", "new\_hot")[20:40]

all\_news\_id = [one["new\_id"] for one in all\_news]

all\_news\_hot\_value = {one["new\_id"]: one["new\_hot"]

for one in all\_news}

print(all\_news\_hot\_value)

#返回热度榜单数据

return new.objects.filter(new\_id\_\_in=all\_news\_id),

all\_news\_hot\_value

11.3.6 前端界面实现

1.运行逻辑

#选择用户登录

def login(request):

if request.method == "GET":

result = dict()

result["users"]=ALLOW\_USERS

result["tags"]=ALLOW\_TAGS

return JsonResponse(result)

elif request.method == "POST":

#从前端获取用户名并写入 session

uname = request.POST.get('username')

request.session["username"]=uname

#前端将标签以逗号拼接的字符串形式返回

tags= request.POST.get('tags')

return JsonResponse({"username": uname,

"tags": tags,"baseclick":0 , "code": 1})

#主页

def home(request):

#从前端请求中获取cate

\_cate = request.GET.get("cateid")

if "username" not in request.session.keys():

return JsonResponse({ "code":0 })

total = 0 #总页数

#如果cate 是推荐页面，走该部分逻辑tag\_flag = 0表示不是从标签召回数据

if \_cate == "1":

news, news\_hot\_value = getRecNews(request)

#如果cate 是热度榜，走该部分逻辑

elif \_cate == "2":

news,news\_hot\_value = getHotNews()

#其他正常的请求获取

else:

\_page\_id = int(request.GET.get("pageid"))

news = new.objects.filter(new\_cate=\_cate).order\_by("-new\_time")

total = news.\_\_len\_\_()

news = news[\_page\_id \* 10:(\_page\_id+1) \* 10]

#切换用户

def switchuser(request):

if "username" in request.session.keys():

uname = request.session["username"]

#删除新闻浏览表中的记录

newbrowse.objects.filter(user\_name=uname).delete()

print("删除用户: %s 的新闻浏览记录 ..." % uname)

#删除session值

del request.session["username"]

print("用户: %s 执行了切换用户动作，删除其对应的session值 ..." % uname)

return JsonResponse({"code":1})

#return HttpResponseRedirect("/index/login/")

2.前端界面的数据配置

#数据库

#mysql配置

DB\_HOST = "127.0.0.1"

DB\_PORT = 3306

DB\_USER = "root"

DB\_PASSWD = "12345678"

DB\_NAME = "newsrec"

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.mysql',

'NAME': DB\_NAME,

'USER': DB\_USER,

'PASSWORD': DB\_PASSWD,

'HOST': DB\_HOST,

'PORT': DB\_PORT

}

}

#密码验证

AUTH\_PASSWORD\_VALIDATORS = [

{

'NAME': 'django.contrib.auth.password\_validation.UserAttributeSimilarityValidator',

},

{

'NAME': 'django.contrib.auth.password\_validation.MinimumLengthValidator',

},

{

'NAME': 'django.contrib.auth.password\_validation.CommonPasswordValidator',

},

{

'NAME': 'django.contrib.auth.password\_validation.NumericPasswordValidator',

},

]

#配置可使用的用户，以便完善整个界面的应用演示

ALLOW\_USERS = ["张三","李四","王五"]

#配置选择用户进入下一页可被显示的标签

ALLOW\_TAGS = ["峰会","AI","技术","百度","互联网","金融","旅游","扶贫","改革开放","战区","公益","中国","脱贫","经济","慈善","文化","文学","国风","音乐","综艺","101"]

3.前端界面配置

import Vue from 'vue'

import App from './App'

import router from './router'

import animate from 'animate.css'

import './assets/style/common.less'

import commontool from './assets/js/tool'

import store from './store'

import layer from 'vue-layer'

Vue.prototype.$layer = layer(Vue)

Vue.use(commontool)

Vue.config.productionTip = false

new Vue({

el: '#app',

router,

store,

components: { App },

template: '<App/>'

})

//此处为“主页（Home）”、“新闻页面（News）”、“登陆页面（Login）”三种页面提供了路由

import Vue from 'vue'

import Router from 'vue-router'

import store from '../store'

import home from '@/pages/Home'

import news from '@/pages/News'

import login from '@/pages/Login'

Vue.use(Router)

const router = new Router({

routes: [

{

path: '/',

name: 'home',

component: home,

meta: {

needLogin: true

}

},

{

path: '/news',

name: 'news',

component: news,

meta: {

needLogin: true

}

},

{

path: '/login',

name: 'login',

component: login,

meta: {

needLogin: false

}

}

]

})

router.beforeEach((to, from, next) => {

if (to.meta.needLogin) {

if (store.state.vuexlogin.isLogin || localStorage.getItem('username')) {

next()

} else {

next({

path: '/login',

query: {redirect: to.fullPath}

})

}

} else {

next()

}

})

export default router

#JavaScript语言三种Vue构架（Home.vue,Login.vue,News.vue）

#前端是一个网页界面，用到了HTML语言。主要涉及一点界面属性（例如界面文字编码格式）的配置

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width,initial-scale=1.0">

<title>Recommon</title>

<link href="./static/style/reset.css" rel="stylesheet" />

</head>

<body style="margin:0">

<div id="app"></div>

</body>

</html>

项目12 口红色号检测推荐系统

12.3 模块实现

12.3.1 数据预处理

1.源数据的存储

2. 处理数据

本部分代码因需要处理多个品牌，代码结构相似，存在重复现象，本处以口红品牌M.A.C为例展示处理方法，相关代码如下：

#-\*- coding: utf-8 -\*-

import json

import re

import os

path="D:/homework/大三下信息系统设计/lipsticks/"

brand="mac/"

brandpathli=[]

dict\_Brand={"name":brand[:-1],"series":None}

dict\_series={"name":None,"lipsticks":None}

dict\_name={"color":"#ffffff", "id":"-1", "name":"none"}

#预设参数，在处理不同品牌时，只需要改变brand和path中的内容即可

def par\_mac(str):

#对该口红的品牌、颜色、色号进行匹配

dict\_name={"color":"#ffffff", "id":"-1", "name":"none"}

#设定默认值，当发现颜色是黑色或ID=-1可及时丢弃不合法的列表

matchObj\_color = re.search( r'#\w{6}', str, re.M|re.I)

if (matchObj\_color!=None):

dict\_name["color"]=matchObj\_color.group().upper()

#print (matchObj\_color.group())调试代码

matchObj\_id = re.search( r"\">\d{3}" , str, re.M|re.I)

if (matchObj\_id!=None):

dict\_name["id"]=matchObj\_id.group()[2:]

#print (matchObj\_id.group()[2:])调试代码

matchObj\_name = re.search( r">\d{0,}[ ]?([A-Z][a-z]\*[,]?[ ]?[!?]?){1,}" , str, re.M|re.I)

if (matchObj\_name!=None):

dict\_name["name"]=matchObj\_name.group()[1:]

return dict\_name

#查找文件夹内所有文件的名称

def eachFile(filepath):

pathDir = os.listdir(filepath)

for allDir in pathDir:

child = os.path.join('%s%s' % (filepath, allDir))

print("children",child)#即该目录下所有文件的名字

brandpathli.append(child)

#读取文件路径名对应的内容

def readFile(filename):

fopen = open(filename, 'r') #r 代表read

list\_series=[]#文件中每一行的内容

for eachLine in fopen:

#print( "读取到得内容如下：",eachLine)调试代码

dict\_name=par\_mac(eachLine)

if(dict\_name["color"]=="#ffffff"):

continue

list\_series.append(dict\_name)

fopen.close()

return list\_series#返回每个系列的所有口红

if \_\_name\_\_ == '\_\_main\_\_':

#filePath = path+brand+subpath调试代码，可以通过不同方式访问数据

filePathC = path+brand

eachFile(filePathC)

list\_brand=[]

for i in brandpathli:

dict\_series={"name":None,"lipsticks":None}

list\_series=readFile(i) #每个系列的所有口红

#print(dict\_name)调试代码

brandname=i.split('/')[-1].split('.')[0]

#print("mainpring",brandname)调试代码

dict\_series["name"]=brandname

dict\_series["lipsticks"]=list\_series

#获得{系列名，口红色号}字典

list\_brand.append(dict\_series)

print("dict\_series",list\_brand)

dict\_Brand["series"]=list\_brand

#dict\_Brand["brands"][1]["series"]调试代码

file = open('D:/homework/大三下信息系统设计/json/'+brand.split('/')[0]+'.json','w',encoding='utf-8')

#在遇到长文本时需要处理编码格式以保证存入的数据不是乱码

json = json.dump(dict\_Brand,file)

print(json)

file.close()

12.3.2 系统搭建

1.人脸识别

在识别唇部之前，定位到人脸，用face\_recognition对图片中进行人脸识别。

import face\_recognition

path="\*.jpg"

#放入一张照片

image = face\_recognition.load\_image\_file(path)

#找到图片中所有人脸

print(type(image))

face\_locations = face\_recognition.face\_locations(image)

print(face\_locations)

#或者在图像中找到面部特征（还可以通过len(face\_locations)得到图片中的人脸数

face\_landmarks\_list = face\_recognition.face\_landmarks(image)

print(face\_landmarks\_list)

#为图像中的每个人脸获取face\_encodings

list\_of\_face\_encodings = face\_recognition.face\_encodings(image)

2.提取唇部轮廓并创建蒙版

pil\_image = Image.fromarray(image)

a=pil\_image.size

#改变类型

print(a,type(a))

blank\_mouse = Image.new('RGB', (a[0],a[1]), (0, 0, 0))

#新建一张黑色图片

for face\_landmarks in face\_landmarks\_list:

    d = ImageDraw.Draw(blank\_mouse, 'RGBA')

    #把嘴唇涂白

    d.polygon(face\_landmarks['top\_lip'], fill=(255,255,255,255))

    d.polygon(face\_landmarks['bottom\_lip'], fill=(255, 255,255, 255))

    d.line(face\_landmarks['top\_lip'], fill=(255, 255,255, 255), width=8)

blank\_mouse.show()

#展示

blank\_mouse=blank\_mouse.crop((pos[3], pos[0], pos[1], pos[2]))

#(left, upper, right, lower) 修剪面部区域

Image.\_show(blank\_mouse)

3.划分嘴唇区域

def masklayer(origin,mask):

#创建函数，origin和mask均为Image函数对应的图片格式，将其转换为数组，再操作

    mask1=np.array(mask)

    print(mask1.shape)

    origin1=np.array(origin)

    print(origin1.shape)

    for i in range(len(mask1)):

        for j in range(len(mask1[i])):

        #如果mask在这个区域是黑色的，则程序不需要这部分图片内容设定原图中该区域rgb=000

if (mask1[i][j][1]>=128):#保留嘴唇区域

                pass

             else:#其他区域变为黑色

                origin1[i][j]=[0,0,0]

    new\_png = Image.fromarray(origin1)

    new\_png.show()

    #new\_png.save('testpic.JPG')

return new\_png

#调用编写的函数，将嘴唇区域框选出来

cropped=masklayer(cropped\_face,blank\_mouse)

4.提取图片颜色

import colorsys

import PIL.Image as Image

def get\_dominant\_color(image):

    #颜色模式转换，以便输出RGB颜色值

    image = image.convert('RGB')

    #生成缩略图，减少计算量，减小cpu压力，缩短运算时间

    image.thumbnail((200, 200))

    max\_score =0

    dominant\_color = 0

    for count,(r,g,b) in image.getcolors(image.size[0]\*image.size[1]):

        #忽略黑色背景

        if (r<100) :

            continue

        #转换为HSV获取颜色饱和度，范围（0,1）

        saturation = colorsys.rgb\_to\_hsv(r/255.0, g/255.0, b/255.0)[1]

        #转换为YUV计算亮度

        y = min(abs(r\*2104+g\*4130+b\*802+4096+131072)>>13,235)

        #将亮度从（16，235）缩放到（0,1）

        y = (y-16.0)/(235-16)

        #忽略高亮色

        if y > 0.9:

            continue

        #选择饱和度高的颜色

        #将饱和度加0.1，这样就不会通过将计数乘以0来完全忽略灰度颜色，但给它们较低的权重

        score = (saturation+0.1)\*count

        if score > max\_score:

            max\_score = score

            dominant\_color = (r,g,b)

    return dominant\_color

5.获取色号库

def RGBhex\_2RGB(rgb\_hex):

#print(rgb\_hex) 调试代码

RGB=[0,0,0]

temp\_num=0

for i in range(len(rgb\_hex)):

temp\_num=0

temp=rgb\_hex[i]

if(i!=0):

#将字母转换为ASCII表中位置

if(temp>='A'and temp<='F'):

temp\_num=ord(temp)-55

#将ABCDEF转换为10~15间的数字

else:

#将字符数字转换为0~9间的数字

temp\_num=ord(temp)-48

if(i%2==1):

#根据位置乘进制

RGB[int((i/2)-0.5)]=RGB[int((i/2)-0.5)]+16\*temp\_num

else:

RGB[int((i/2)-0.5)]=RGB[int((i/2)-0.5)]+temp\_num

#print(RGB)调试代码

return RGB

import numpy as np

import json

#对色号库进行操作

#将品牌、系列颜色ID和颜色名称保存到汇总列表中

def operate(target\_color):

sum\_all=0

with open('lipstick.json', 'r', encoding='utf-8') as f:

js2dic = json.load(f)

#读取json

brands\_n=len(js2dic['brands'])

print(brands\_n)

series\_n=0

for brands\_i in range(brands\_n):

series\_n=len(js2dic['brands'][brands\_i]['series'])

print("{0} has {1} series".format((js2dic['brands'][brands\_i]['name']),series\_n))

for series\_i in range(series\_n):

brand\_name=js2dic['brands'][brands\_i]['name']

lip\_name=js2dic['brands'][brands\_i]['series'][series\_i]['name']

color\_num=len(js2dic['brands'][brands\_i]['series'][series\_i]['lipsticks'])

sum\_all=color\_num+sum\_all

#计算颜色总数

print(sum\_all)

catalog=np.zeros((sum\_all,4), dtype=(str,20))

catalog\_color=np.zeros((sum\_all,3), dtype=int)

#根据颜色数分配空间

rank\_color=np.zeros((sum\_all,1),dtype=int)

6.比较并得出结果

#catalog分为四部分：品牌名称、唇膏名称、色号ID、色号值

#将信息存入表格，这个循环可以将信息存入Python程序建立数组中

sum\_i=0

for brands\_i in range(brands\_n):

series\_n=len(js2dic['brands'][brands\_i]['series'])

#print("brand\_name",js2dic['brands'][brands\_i]['name'])

catalog[sum\_i][0]=js2dic['brands'][brands\_i]['name']

for series\_i in range(series\_n): color\_num=len(js2dic['brands'][brands\_i]['series'][series\_i]['lipsticks'])

for color\_i in range(color\_num):

catalog[sum\_i][0]=js2dic['brands'][brands\_i]['name'] catalog[sum\_i][1]=js2dic['brands'][brands\_i]['series'][series\_i]['name'] catalog[sum\_i][2]=js2dic['brands'][brands\_i]['series'][series\_i]['lipsticks'][color\_i]['name'] catalog[sum\_i][3]=js2dic['brands'][brands\_i]['series'][series\_i]['lipsticks'][color\_i]['id'] catalog\_color[sum\_i]=RGBhex\_2RGB(js2dic['brands'][brands\_i]['series'][series\_i]['lipsticks'][color\_i]['color'])

sum\_i+=1

#print(sum\_i)调试代码

print(catalog.shape)

RGB\_distance=np.zeros((sum\_all,1), dtype=float)

for i in range(sum\_all):

#计算相似度，target是此前通过domain得到的值

RGB\_distance[i]=abs(target\_color[0]-catalog\_color[i][0])+abs((target\_color[1]-catalog\_color[i][1])\*(1/5))+abs(target\_color[2]-catalog\_color[i][2])

RGB\_distance.tolist()

result=sorted(range(len(RGB\_distance)), key=lambda k: RGB\_distance[k])

#获得颜色最像的三只口红（以颜色的相近度为规则排序，返回位置数据）

print("颜色最像的三只口红及其颜色")

result\_show=[]

for i in range(3):

loc=result[i]

color\_show=tuple(catalog\_color[loc])

print("catalog index",catalog[loc],color\_show)

#operate([155, 44, 69]) 调试代码

if \_\_name\_\_ == '\_\_main\_\_':

get=get\_dominant\_color(cropped)

print("获得的口红颜色{0}".format(get))

#计算相似度，get是此前通过domain得到的值

operate(get)

7.创建图形化界面

#-\*- coding: utf-8 -\*-

#从读取ui文件“myuidesign.ui”生成的窗体实现

#创建人：PyQt5 UI代码生成器5.10

#警告！此文件中做的所有更改都将丢失

from PyQt5 import QtCore, QtGUI, QtWidgets

from PyQt5.QtWidgets import QFileDialog, QWidget,QGraphicsScene

from PyQt5.QtCore import QFileInfo

from detectface import my\_face\_recognition

try:

    \_fromUtf8 = QtCore.QString.fromUtf8

except AttributeError:

    def \_fromUtf8(s):

        return s

try:

    \_encoding = QtGUI.QApplication.UnicodeUTF8

    def \_translate(context, text, disambig):

        return QtWidgets.QApplication.translate(context, text, disambig, \_encoding)

except AttributeError:

    def \_translate(context, text, disambig):

        return QtWidgets.QApplication.translate(context, text, disambig)

#以上代码处理了文字的编码格式

#建立UI函数

class Ui\_Dialog(object):

    def \_\_init\_\_(self):

        super(Ui\_Dialog, self).\_\_init\_\_()

        self.imgPath=""

        self.face\_recognize\_object =None

        self.showFullImage = True

        self.brand=[]

        self.color=[]

    def setupUi(self, Form): #设置界面

        Form.setObjectName("Form")

        Form.resize(817, 630)

        self.horizontalLayoutWidget = QtWidgets.QWidget(Form)

        self.horizontalLayoutWidget.setGeometry(QtCore.QRect(10,20,801,611))

        self.horizontalLayoutWidget.setObjectName("horizontalLayoutWidget")

        self.horizontalLayout = QtWidgets.QHBoxLayout(self.horizontalLayoutWidget)

        self.horizontalLayout.setContentsMargins(0, 0, 0, 0)

        self.horizontalLayout.setObjectName("horizontalLayout")

        self.verticalLayout\_3 = QtWidgets.QVBoxLayout()

        self.verticalLayout\_3.setObjectName("verticalLayout\_3")

        self.label = QtWidgets.QLabel(self.horizontalLayoutWidget)

        self.label.setMinimumSize(QtCore.QSize(330, 50))

font = QtGUI.QFont()

        font.setFamily("汉仪唐美人W")

        font.setPointSize(36)

        font.setUnderline(False)

        self.label.setFont(font)

        self.label.setWordWrap(False)

        self.label.setObjectName("label")

        self.verticalLayout\_3.addWidget(self.label)

       self.graphicsView=QtWidgets.QGraphicsView(self.horizontalLayoutWidget)

        self.graphicsView.setObjectName("graphicsView")

        self.verticalLayout\_3.addWidget(self.graphicsView)

        self.pushButton = QtWidgets.QPushButton(self.horizontalLayoutWidget)

        font = QtGUI.QFont()

        font.setFamily("汉仪唐美人W")

        self.pushButton.setFont(font)

        self.pushButton.setObjectName("pushButton")

self.pushButton.setMinimumSize(QtCore.QSize(20, 50))

        self.verticalLayout\_3.addWidget(self.pushButton)

        self.horizontalLayout.addLayout(self.verticalLayout\_3)

        self.verticalLayout\_4 = QtWidgets.QVBoxLayout()

        self.verticalLayout\_4.setObjectName("verticalLayout\_4")

        self.label\_2 = QtWidgets.QLabel(self.horizontalLayoutWidget)

        self.label\_2.setMinimumSize(QtCore.QSize(330, 50))

        font = QtGUI.QFont()

        font.setFamily("汉仪唐美人W")

        font.setPointSize(36)

        self.label\_2.setFont(font)

        self.label\_2.setObjectName("label\_2")

        self.verticalLayout\_4.addWidget(self.label\_2)

       self.textBrowser= QtWidgets.QTextBrowser(self.horizontalLayoutWidget)

        self.textBrowser.setObjectName("textBrowser")

        self.verticalLayout\_4.addWidget(self.textBrowser)

       self.pushButton\_2= QtWidgets.QPushButton(self.horizontalLayoutWidget)

        font = QtGUI.QFont()

        font.setFamily("汉仪唐美人W")

        self.pushButton\_2.setFont(font)

        self.pushButton\_2.setObjectName("pushButton\_2")

self.pushButton\_2.setMinimumSize(QtCore.QSize(20, 50))

        self.verticalLayout\_4.addWidget(self.pushButton\_2)

        self.horizontalLayout.addLayout(self.verticalLayout\_4)

        self.retranslateUi(Form)

        self.pushButton.clicked.connect(self.on\_pushButton\_clicked)

        self.pushButton\_2.clicked.connect(self.on\_pushButton\_2\_clicked)

        QtCore.QMetaObject.connectSlotsByName(Form)

def on\_pushButton\_clicked(self):

        #get the image path and show it in the view

        self.face\_recognize\_object = my\_face\_recognition()

        fileName, filetype = QFileDialog.getOpenFileName(None, "选择文件", r"此处需要填写文件路径的起点", "Images (\*.png \*.jpg)")

        self.imgPath=fileName

        print (self.imgPath)

        if self.imgPath != '':

            self.face\_recognize\_object.operates\_(self.imgPath)

            if self.showFullImage == True:

                self.face\_recognize\_object.showImg('original')

            scene = QGraphicsScene() #创建场景

            pixmap = QtGUI.QPixmap(self.imgPath)

#调用QtGUI.QPixmap方法，打开一个图片，存放在变量中

            scene.addItem(QtWidgets.QGraphicsPixmapItem(pixmap))

#添加图片到场景中

            self.graphicsView.setScene(scene)

#将场景添加到graphicsView中

            self.graphicsView.show()

#显示

            self.textBrowser.clear()

            self.textBrowser.append(str(self.face\_recognize\_object.errdet))

#输出图片是否合法

def on\_pushButton\_2\_clicked(self):

        self.face\_recognize\_object.AI()

        self.brand=self.face\_recognize\_object.register\_lps

        self.color=self.face\_recognize\_object.register\_rgb

        print(self.brand)

        self.textBrowser.clear()

        self.textBrowser.append(str(self.face\_recognize\_object.errdet))

#输出错误信息，若图片不合法则在引用的函数返回为空。若图片合法则会继续输出，看到口红信息

       strout="最像您输入口红的三只色号库内口红分别为!"

flag=0 #设定flag用于检测是否非法

for i in range(len(self.brand)):

flag=1 #如果是合法输入，那么flag就会置1，后续不会提示错误

strout=strout+'\n'

print("flag1")

for j in range(len(self.brand[i])):

if(self.brand[i][j]!='none'):

strout=strout+str(self.brand[i][j])

self.textBrowser.append(strout)

if(flag==0):

self.textBrowser.clear()

self.textBrowser.append("这张图片不可以进行处理，请换一张吧")

print("flag0")

   #一些有关UI的固定操作，来自ui->py文件的转换器

def retranslateUi(self, Form):

        \_translate = QtCore.QCoreApplication.translate

        Form.setWindowTitle(\_translate("Form", "Form"))

        self.label.setText(\_translate("Form", " Original pic"))

        self.pushButton.setText(\_translate("Form", "选择文件"))

        self.label\_2.setText(\_translate("Form", "     result"))

        self.pushButton\_2.setText(\_translate("Form", "开始识别"))

if \_\_name\_\_ == "\_\_main\_\_":

    import sys

    app = QtWidgets.QApplication(sys.argv)

    Dialog = QtWidgets.QDialog()

    ui = Ui\_Dialog()

    ui.setupUi(Dialog)

    Dialog.show()

    sys.exit(app.exec\_())

8.将流程封装为类和函数

def operates\_(self,input\_path):

        self.imgPath=input\_path #展示图片

    def AI(self):

        self.load\_pic()

        self.get=self.get\_dominant\_color(self.resultImg)

       print("the extracted RGB value of the color is {0}".format(self.get))

        self.data\_operate()

    def errordetect(self): #错误处理

        image = face\_recognition.load\_image\_file(self.imgPath)

        face\_locations = face\_recognition.face\_locations(image)

        if(len(face\_locations)!=1): #不是仅有一张人脸在图片中

            if(len(face\_locations)==0): #未检测到

                self.errdet="more/less than one face in the pic"

                return "can't find people"

else：#不只一个人

                self.errdet="more/less than one face in the pic"

            return "more/less than one face in the pic"

        else:

            return "DEAL"

9.对GUI的显示效果进行美化

from QCandyUi import CandyWindow

#调整按钮的大小使之更清晰可见

self.pushButton.setMinimumSize(QtCore.QSize(20, 50))

self.pushButton\_2.setMinimumSize(QtCore.QSize(20, 50))

#改变定义窗口类的调用方式

if \_\_name\_\_ == "\_\_main\_\_":

import sys

app = QtWidgets.QApplication(sys.argv)

Dialog = QtWidgets.QDialog()

ui = Ui\_Dialog()

ui.setupUi(Dialog)

Dialog = CandyWindow.createWindow(Dialog, 'pink') #增加，使QCandyUi运行

Dialog.show()

sys.exit(app.exec\_())

项目13 基于矩阵分解算法的Steam游戏推荐系统

13.3 模块实现

13.3.1 数据预处理

相关代码如下：

import numpy as np

import pandas as pd

import tensorflow.compat.v1 as tf

tf.disable\_v2\_behavior()

import random

from collections import Counter

from sklearn.metrics import roc\_curve, auc, average\_precision\_score

import joblib

#导入数据集并列表显示

path = './steam-video-games/steam-200k.csv'

df = pd.read\_csv(path, header = None, names = ['UserID', 'Game', 'Action', 'Hours', 'Not Needed'])

df.head()

相关代码如下：

#从购买记录和游玩记录中筛选出游戏时长

df['Hours\_Played'] = df['Hours'].astype('float32')

df.loc[(df['Action']=='purchase')&(df['Hours']==1.0), 'Hours\_Played'] = 0

#排序

df.UserID = df.UserID.astype('int')

df = df.sort\_values(['UserID', 'Game', 'Hours\_Played'])

#整理为新的表格clean\_df

clean\_df = df.drop\_duplicates(['UserID', 'Game'], keep = 'last').drop(['Action', 'Hours', 'Not Needed'], axis = 1)

clean\_df.head()

#输出数据集中的用户数量和游戏数量

n\_users = len(clean\_df.UserID.unique())

n\_games = len(clean\_df.Game.unique())

print('用户-游戏数据集中一共有{0}个用户，{1}个游戏'.format(n\_users, n\_games))

由于是稀疏矩阵，因而使用矩阵分解算法可以得到较好的效果，相关代码如下：

#计算矩阵的稀疏程度

sparsity = clean\_df.shape[0] / float(n\_users \* n\_games)

print('用户-游戏矩阵中有效数据占比为：{:.2%}'.format(sparsity))

#序列化ID相关代码

#建立序列化的ID，方便使用

#用户ID到用户序列化ID的字典

user2idx = {user: i for i, user in enumerate(clean\_df.UserID.unique())}

#用户序列化ID到用户ID的字典

idx2user = {i: user for user, i in user2idx.items()}

#游戏名到游戏序列化ID的字典

game2idx = {game: i for i, game in enumerate(clean\_df.Game.unique())}

#游戏序列化ID到游戏名的字典

idx2game = {i: game for game, i in game2idx.items()}

#将字典保存，用于PyQt5中

joblib.dump(idx2game, './Save\_data/idx2game.pkl')

joblib.dump(game2idx, './Save\_data/game2idx.pkl')

将用户ID、游戏名称、游戏时长分别存储为数组，其中用户ID、游戏名称使用前一步得到的序列化ID存储，以便使用，相关代码如下：

#用户序列化ID-游戏序列化ID-游戏时长

user\_idx = clean\_df['UserID'].apply(lambda x: user2idx[x]).values

game\_idx = clean\_df['gamesIdx'] = clean\_df['Game'].apply(lambda x:game2idx[x]).values

hours = clean\_df['Hours\_Played'].values

#保存游戏时长矩阵

hours\_save = np.zeros(shape = (n\_users, n\_games))

for i in range(len(user\_idx)):

hours\_save[user\_idx[i], game\_idx[i]] = hours[i]

joblib.dump(hours\_save, './Save\_data/hours.pkl')

相关代码如下：

#建立稀疏矩阵存储大数据集

#购买矩阵

#未购买标识为0

#购买标识为1

#置信度矩阵

#根据游戏时长提高置信度，最低为1

zero\_matrix = np.zeros(shape = (n\_users, n\_games))

#购买矩阵

user\_game\_pref = zero\_matrix.copy()

user\_game\_pref[user\_idx, game\_idx] = 1

#保存购买矩阵

joblib.dump(user\_game\_pref, './Save\_data/buy.pkl')

#置信度矩阵

user\_game\_interactions = zero\_matrix.copy()

user\_game\_interactions[user\_idx, game\_idx] = hours + 1

#为保证准确率，需要用户购买的数量达到一定值，设置阈值为10款游戏

k = 5

#对于每个用户计算他们购买的游戏数量

purchase\_counts = np.apply\_along\_axis(np.bincount, 1, user\_game\_pref.astype(int))

buyers\_idx = np.where(purchase\_counts[:, 1] >= 2 \* k)[0]

#购买超过2\*k个游戏的买家集合

print('{0}名玩家购买了至少{1}款游戏'.format(len(buyers\_idx), 2 \* k))

#保存有效购买用户名单

joblib.dump(buyers\_idx, './Save\_data/buyers.pkl')

在2189名用户中，划分出训练集、测试集、验证集，比例分别为80%、10%、10%，相关代码如下：

test\_frac = 0.2 #10%数据用来验证，10%数据用来测试

test\_users\_idx = np.random.choice(buyers\_idx,

size = int(np.ceil(len(buyers\_idx) \* test\_frac)),

replace = False)

val\_users\_idx = test\_users\_idx[:int(len(test\_users\_idx) / 2)]

test\_users\_idx = test\_users\_idx[int(len(test\_users\_idx) / 2):]

准确率的计算方式：通过掩盖5个用户购买的游戏，使用模型得到推荐的5个游戏与掩盖的游戏相比计算正确率，相关代码如下：

#在训练集中掩盖k个游戏

def data\_process(dat, train, test, user\_idx, k):

for user in user\_idx:

purchases = np.where(dat[user, :] == 1)[0]

mask = np.random.choice(purchases, size = k, replace = False)

train[user, mask] = 0

test[user, mask] = dat[user, mask]

return train, test

train\_matrix = user\_game\_pref.copy()

test\_matrix = zero\_matrix.copy()

val\_matrix = zero\_matrix.copy()

train\_matrix, val\_matrix = data\_process(user\_game\_pref, train\_matrix,

val\_matrix, val\_users\_idx, k)

train\_matrix, test\_matrix = data\_process(user\_game\_pref, train\_matrix,

test\_matrix, test\_users\_idx, k)

#测试是否将部分游戏掩盖

test\_matrix[test\_users\_idx[0],test\_matrix[test\_users\_idx[0],:].nonzero()[0]]

train\_matrix[test\_users\_idx[0],test\_matrix[test\_users\_idx[0],:].nonzero()[0]]

13.3.2 模型构建

数据加载进模型之后，需要定义模型结构，并优化损失函数。

1.定义模型结构

使用矩阵分解算法，将用户—游戏这稀疏矩阵用两个小矩阵——特征—游戏矩阵和用户—特征矩阵，进行近似替代。

tf.reset\_default\_graph()

#偏好矩阵

pref = tf.placeholder(tf.float32, (n\_users, n\_games))

#游戏时间矩阵

interactions = tf.placeholder(tf.float32, (n\_users, n\_games))

user\_idx = tf.placeholder(tf.int32, (None))

n\_features = 30 #隐藏特征个数设置为30

#X矩阵（用户-隐藏特征）表示用户潜在偏好

X = tf.Variable(tf.truncated\_normal([n\_users, n\_features], mean = 0,

stddev = 0.05), dtype = tf.float32, name = 'X')

#Y矩阵（游戏-隐藏特征）表示游戏潜在特征

Y = tf.Variable(tf.truncated\_normal([n\_games, n\_features], mean = 0,

stddev = 0.05), dtype = tf.float32, name = 'Y')

#初始化用户偏差

user\_bias = tf.Variable(tf.truncated\_normal([n\_users, 1], stddev = 0.2))

#将向量连接到用户矩阵

X\_plus\_bias = tf.concat([X,

user\_bias,

tf.ones((n\_users, 1), dtype = tf.float32)],

axis = 1)

#初始化游戏偏差

item\_bias = tf.Variable(tf.truncated\_normal([n\_games, 1], stddev = 0.2))

#将向量连接到游戏矩阵

Y\_plus\_bias = tf.concat([Y,

tf.ones((n\_games, 1), dtype = tf.float32),

item\_bias],

axis = 1)

#通过矩阵乘积确定结果评分矩阵

pred\_pref = tf.matmul(X\_plus\_bias, Y\_plus\_bias, transpose\_b = True)

#使用游戏时长与alpha参数构造置信度矩阵

conf = 1 + conf\_alpha \* interactions

2.优化损失函数

相关代码如下：

cost = tf.reduce\_sum(tf.multiply(conf, tf.square(tf.subtract(pref, pred\_pref))))

l2\_sqr = tf.nn.l2\_loss(X) + tf.nn.l2\_loss(Y) + tf.nn.l2\_loss(user\_bias) + tf.nn.l2\_loss(item\_bias)

lambda\_c = 0.01

loss = cost + lambda\_c \* l2\_sqr

lr = 0.05

optimize = tf.train.AdagradOptimizer(learning\_rate = lr).minimize(loss)

13.3.3 模型训练及保存

相关代码如下：

#精确度计算优化，将游戏本体和DLC合并为同一种游戏

def precision\_dlc(recommandations, labels):

#推荐的游戏按单词划分

recommandations\_split = []

#实际购买的游戏按单词划分

labels\_split = []

for label in labels:

labels\_split.append(idx2game[label].split())

for game in recommandations:

recommandations\_split.append(idx2game[game].split())

count = 0

for game in recommandations\_split:

for label in labels\_split:

#当推荐的游戏与实际购买的游戏单词重合度高于阈值判定为同一款游戏

if(len(set(game)&set(label))/min(len(game),len(label))) > 0.2:

count += 1

break

return float(count / len(recommandations))

推荐的游戏方式为，对用户—游戏矩阵进行排序，选取评分最高的5个游戏作为推荐，计算准确率并返回，相关代码如下：

#从预测的列表中挑选最高的k个

def top\_k\_precision(pred, mat, k, user\_idx):

precisions = []

for user in user\_idx:

rec = np.argsort(-pred[user, :])

#选取推荐评分最高的k个

top\_k = rec[:k]

labels = mat[user, :].nonzero()[0]

#计算推荐与实际的准确率并返回

precision = precision\_dlc(top\_k, labels)

precisions.append(precision)

return np.mean(precisions)

1.模型训练

相关代码如下：

iterations = 500

#绘图用数据：误差、训练集准确率

fig\_loss = np.zeros([iterations])

fig\_train\_precision = np.zeros([iterations])

with tf.Session() as sess:

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

for i in range(iterations):

sess.run(optimize, feed\_dict = {pref: train\_matrix,

interactions: user\_game\_interactions})

if i % 10 == 0:

mod\_loss = sess.run(loss, feed\_dict = {pref: train\_matrix,

interactions: user\_game\_interactions})

mod\_pred = pred\_pref.eval()

train\_precision = top\_k\_precision(mod\_pred, train\_matrix,

k, val\_users\_idx)

val\_precision = top\_k\_precision(mod\_pred, val\_matrix,

k, val\_users\_idx)

print('当前进度：{}...'.format(i),

'误差为：{:.2f}...'.format(mod\_loss),

'训练集上的正确率：{:.3f}...'.format(train\_precision),

'验证集上的正确率：{:.3f}'.format(val\_precision))

fig\_loss[i] = sess.run(loss, feed\_dict = {pref: train\_matrix,

interactions: user\_game\_interactions})

fig\_train\_precision[i] = top\_k\_precision(mod\_pred, train\_matrix,

k, val\_users\_idx)

rec = pred\_pref.eval()

test\_precision = top\_k\_precision(rec, test\_matrix, k, test\_users\_idx)

print('\n')

print('模型完成，正确率为：{:.3f}'.format(test\_precision))

2.模型保存

为方便使用模型，需要将训练得到的结果使用Joblib进行保存，相关代码如下：

#将训练得到的评分矩阵保存

with tf.Session() as sess:

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

joblib.dump(pred\_pref.eval(), './Save\_data/rec.pkl')

13.3.4 模型应用

1.制作页面

相关操作如下：

(1)使用代码绘制页面的基础布局，创建Recommandation类。

class Recommandation(QWidget):

#初始化

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.initUI()

#初始化布局

def initUI(self):

#设置界面的初始位置和大小

self.setGeometry(600,200,450,550)

#窗口名

self.setWindowTitle('steam游戏推荐')

#设置组件，以下为标签

self.lb1 = QLabel('请输入游戏名：',self)

#这是所在位置

self.lb1.move(20,20)

self.lb2 = QLabel('请输入游戏名：',self)

self.lb2.move(20,80)

self.lb3 = QLabel('请输入游戏名：',self)

self.lb3.move(20,140)

self.lb4 = QLabel('请输入游戏名：',self)

self.lb4.move(20,200)

self.lb5 = QLabel('请输入游戏名：',self)

self.lb5.move(20,260)

#以下为下拉输入框的创建

self.combobox1 = QComboBox(self, minimumWidth=200)

self.combobox1.move(100,20)

self.combobox1.setEditable(True)

self.combobox2 = QComboBox(self, minimumWidth=200)

self.combobox2.move(100,80)

self.combobox2.setEditable(True)

self.combobox3 = QComboBox(self, minimumWidth=200)

self.combobox3.move(100,140)

self.combobox3.setEditable(True)

self.combobox4 = QComboBox(self, minimumWidth=200)

self.combobox4.move(100,200)

self.combobox4.setEditable(True)

self.combobox5 = QComboBox(self, minimumWidth=200)

self.combobox5.move(100,260)

self.combobox5.setEditable(True)

#以下为输入的按键设置

self.bt1 = QPushButton('请输入游戏时间',self)

self.bt1.move(330,20)

self.bt2 = QPushButton('请输入游戏时间',self)

self.bt2.move(330,80)

self.bt3 = QPushButton('请输入游戏时间',self)

self.bt3.move(330,140)

self.bt4 = QPushButton('请输入游戏时间',self)

self.bt4.move(330,200)

self.bt5 = QPushButton('请输入游戏时间',self)

self.bt5.move(330,260)

#推荐按钮

self.bt=QPushButton('推荐开始',self)

self.bt.move(20,400)

#初始化下拉输入框

self.init\_combobox()

#连接按键与槽

self.bt1.clicked.connect(self.timeDialog)

self.bt2.clicked.connect(self.timeDialog)

self.bt3.clicked.connect(self.timeDialog)

self.bt4.clicked.connect(self.timeDialog)

self.bt5.clicked.connect(self.timeDialog)

#连接推荐

self.bt.clicked.connect(self.recommand)

connect()是Qt特有的信号与槽机制，槽接收到信号进行处理。在这里使用了clicked 作为信号，单击按键会发出信号。

(2)初始化下拉输入框，将gamelist输入进下拉框的菜单，以及添加自动补全机能。

#初始化下拉输入框

def init\_combobox(self):

#增加选项元素

for i in range(len(gamelist)):

self.combobox1.addItem(gamelist[i])

self.combobox2.addItem(gamelist[i])

self.combobox3.addItem(gamelist[i])

self.combobox4.addItem(gamelist[i])

self.combobox5.addItem(gamelist[i])

self.combobox1.setCurrentIndex(-1)

self.combobox2.setCurrentIndex(-1)

self.combobox3.setCurrentIndex(-1)

self.combobox4.setCurrentIndex(-1)

self.combobox5.setCurrentIndex(-1)

#增加自动补全

self.completer = QCompleter(gamelist)

#补全方式

self.completer.setFilterMode(Qt.MatchStartsWith)

self.completer.setCompletionMode(QCompleter.PopupCompletion)

self.combobox1.setCompleter(self.completer)

self.combobox2.setCompleter(self.completer)

self.combobox3.setCompleter(self.completer)

self.combobox4.setCompleter(self.completer)

self.combobox5.setCompleter(self.completer)

(3)设置槽，同时存储数据

相关操作如下

def timeDialog(self):

#获取信号

sender = self.sender()

if sender == self.bt1:

#获取下拉输入框1输入的游戏名

gamename = self.combobox1.currentText()

#通过字典game2idx查询获得的游戏名所对应的序列号

gameid = game2idx.get(gamename)

#没有序列号的情况，可以理解为未输入正确的游戏名，或者输入为空

if gameid == None:

#这种情况下生成一个MessageBox报错

reply = QMessageBox.information(self,'Error','请输入正确的游戏名!', QMessageBox.Close)

else:

#输入正确的情况，将游戏名字、ID，分别记录到一个字典里，方便保存与更改

gamedict[1] = gamename

idxdict[1] = gameid

#弹出一个文本输入框，要求输入对应游戏时长

text, ok = QInputDialog.getDouble(self, '游戏时间', '请输入游戏时间：', min = 0.1)

#如果输入正确，将时长记录到一个字典中，方便保存与更改

if ok:

timedict[1] = text

elif sender == self.bt2:

gamename = self.combobox2.currentText()

gameid = game2idx.get(gamename)

if gameid == None:

reply = QMessageBox.information(self,'Error','请输入正确的游戏名!', QMessageBox.Close)

else:

gamedict[2] = gamename

idxdict[2] = gameid

text, ok = QInputDialog.getDouble(self, '游戏时间', '请输入游戏时间：', min = 0.1)

if ok:

timedict[2] = text

elif sender == self.bt3:

gamename = self.combobox3.currentText()

gameid = game2idx.get(gamename)

if gameid == None:

reply = QMessageBox.information(self,'Error','请输入正确的游戏名!', QMessageBox.Close)

else:

gamedict[3] = gamename

idxdict[3] = gameid

text, ok = QInputDialog.getDouble(self, '游戏时间', '请输入游戏时间：', min = 0.1)

if ok:

timedict[3] = text

elif sender == self.bt4:

gamename = self.combobox4.currentText()

gameid = game2idx.get(gamename)

if gameid == None:

reply = QMessageBox.information(self,'Error','请输入正确的游戏名!', QMessageBox.Close)

else:

gamedict[4] = gamename

idxdict[4] = gameid

text, ok = QInputDialog.getDouble(self, '游戏时间', '请输入游戏时间：', min = 0.1)

if ok:

timedict[4] = text

elif sender == self.bt5:

gamename = self.combobox5.currentText()

gameid = game2idx.get(gamename)

if gameid == None:

reply = QMessageBox.information(self,'Error','请输入正确的游戏名!', QMessageBox.Close)

else:

gamedict[5] = gamename

idxdict[5] = gameid

text, ok = QInputDialog.getDouble(self, '游戏时间', '请输入游戏时间：', min = 0.1)

if ok:

timedict[5] = text

4)验证数据是否输入完毕，以及准备调用模型

def recommand(self):

#验证是否存在没有写入的数据

c = 0

for i in range(1,6):

if gamedict[i] == "NULL":

c+=1

if idxdict[i] == "NULL":

c+=1

if timedict[i] == "NULL":

c+=1

#全部写完的情况

if c == 0:

#将字典转化为列表

usertime = list(timedict.values())

useridx = list(idxdict.values())

#调用模型

allrecidx = UserSimilarity(useridx,usertime)

#降序排列数据

rr = np.argsort(-allrecidx)

#获取排行前五的游戏ID

top\_k = rr[:5]

#将ID对应的游戏名字输入数组

for i in top\_k:

recgame.append(idx2game[i])

#将数组转化为字符串并输出

reclist = ','.join(recgame)

reply = QMessageBox.information(self,'推荐的游戏','给您推荐的游戏是'+reclist, QMessageBox.Close)

#存在没有写完的数据，要求重新写入

else:

reply = QMessageBox.information(self,'Error','请输入全部数据!', QMessageBox.Close)

2.模型导入及调用

相关操作如下：

(1)加载当前文件夹下的Save\_data模型

game2idx = joblib.load('./Save\_data/game2idx.pkl')

idx2game = joblib.load('./Save\_data/idx2game.pkl')

rec = joblib.load('./Save\_data/rec.pkl')

hours = joblib.load('./Save\_data/hours.pkl')

buy = joblib.load('./Save\_data/buy.pkl')

users = joblib.load('./Save\_data/buyers.pkl')

(2)创建一个用户相似度函数，用于刻画Qt里收集到的数据与训练出的用户相似度最高的数据作为输出

def UserSimilarity(games, game\_hours):

similarity = np.zeros(len(users)) # 用户相似度矩阵

for i in range(len(users)):

#计算用户输入的游戏与数据集中每个用户购买游戏的重合度

coincidence = 0 #重合度，每重合一个游戏加1

positions = [] #重合游戏在games中的位置

#获取数据集中的第i个玩家与用户输入的重合情况

for ii in range(len(games)):

if games[ii] in np.where(buy[users[i], :] == 1)[0]:

coincidence += 1

positions.append(ii)

#如果没有重合，则相似度为0，跳过

if coincidence == 0:

continue

simi = []

#将重合的游戏，根据时长和相同游戏的时长差取绝对值，根据e^-x计算出相似度

for position in positions:

game = games[position]

hour = abs(game\_hours[position] - hours[users[i], game])

simi.append(math.exp(-hour))

#对所有相似度取均值，得到用户与数据集中第i个玩家的相似度similarity[i]

similarity[i] = sum(simi) / coincidence

#相似度与玩家—游戏矩阵每一行相乘

for i in range(len(users)):

user = users[i]

rec[user] = rec[user] \* similarity[i]

new\_rec = np.zeros(len(rec[0])) # 1\*n\_games矩阵

#将玩家—游戏矩阵按列相加，得到用户对每个游戏的喜好程度，即new\_rec矩阵

for i in range(len(new\_rec)):

for user in users:

new\_rec[i] += rec[user][int(i)]

return new\_rec

3.模型应用代码

相关代码如下：

import joblib

import numpy as np

import pandas as pd

import math

import sys

from PyQt5.QtWidgets import \*

from PyQt5.QtGui import \*

from PyQt5.QtCore import \*

#读取数据

game2idx = joblib.load('./Save\_data/game2idx.pkl')

idx2game = joblib.load('./Save\_data/idx2game.pkl')

rec = joblib.load('./Save\_data/rec.pkl')

hours = joblib.load('./Save\_data/hours.pkl')

buy = joblib.load('./Save\_data/buy.pkl')

users = joblib.load('./Save\_data/buyers.pkl')

#游戏名称列表

gamelist = list(game2idx)

#游戏数

n\_game = len(gamelist)

#传入字典

gamedict = {1:"NULL",2:"NULL",3:"NULL",4:"NULL",5:"NULL"}

timedict = {1:"NULL",2:"NULL",3:"NULL",4:"NULL",5:"NULL"}

idxdict = {1:"NULL",2:"NULL",3:"NULL",4:"NULL",5:"NULL"}

#下面两个是要传递的

usertime=[]

useridx=[]

#下面是返回的推荐游戏

recgame=[]

#相似度推荐

def UserSimilarity(games, game\_hours):

similarity = np.zeros(len(users)) #用户相似度矩阵

for i in range(len(users)):

#计算用户输入的游戏与数据集中每个用户购买游戏的重合度

coincidence = 0 #重合度

positions = [] #重合游戏在games中的位置

for ii in range(len(games)):

if games[ii] in np.where(buy[users[i], :] == 1)[0]:

coincidence += 1

positions.append(ii)

if coincidence == 0:

continue

simi = []

for position in positions:

game = games[position]

hour = abs(game\_hours[position] - hours[users[i], game])

simi.append(math.exp(-hour))

similarity[i] = sum(simi) / coincidence

#相似度与玩家—游戏矩阵每一行相乘

for i in range(len(users)):

user = users[i]

rec[user] = rec[user] \* similarity[i]

new\_rec = np.zeros(len(rec[0])) #1\*n\_games矩阵

for i in range(len(new\_rec)):

for user in users:

new\_rec[i] += rec[user][int(i)]

return new\_rec

class Recommandation(QWidget):

#初始化

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.initUI()

#初始化布局

def initUI(self):

#设置界面的初始位置和大小

self.setGeometry(600,200,450,550)

#窗口名

self.setWindowTitle('steam游戏推荐')

#设置组件，以下为标签

self.lb1 = QLabel('请输入游戏名：',self)

#这是所在位置

self.lb1.move(20,20)

self.lb2 = QLabel('请输入游戏名：',self)

self.lb2.move(20,80)

self.lb3 = QLabel('请输入游戏名：',self)

self.lb3.move(20,140)

self.lb4 = QLabel('请输入游戏名：',self)

self.lb4.move(20,200)

self.lb5 = QLabel('请输入游戏名：',self)

self.lb5.move(20,260)

#以下为下拉输入框的创建

self.combobox1 = QComboBox(self, minimumWidth=200)

self.combobox1.move(100,20)

self.combobox1.setEditable(True)

self.combobox2 = QComboBox(self, minimumWidth=200)

self.combobox2.move(100,80)

self.combobox2.setEditable(True)

self.combobox3 = QComboBox(self, minimumWidth=200)

self.combobox3.move(100,140)

self.combobox3.setEditable(True)

self.combobox4 = QComboBox(self, minimumWidth=200)

self.combobox4.move(100,200)

self.combobox4.setEditable(True)

self.combobox5 = QComboBox(self, minimumWidth=200)

self.combobox5.move(100,260)

self.combobox5.setEditable(True)

#以下为输入的按键设置

self.bt1 = QPushButton('请输入游戏时间',self)

self.bt1.move(330,20)

self.bt2 = QPushButton('请输入游戏时间',self)

self.bt2.move(330,80)

self.bt3 = QPushButton('请输入游戏时间',self)

self.bt3.move(330,140)

self.bt4 = QPushButton('请输入游戏时间',self)

self.bt4.move(330,200)

self.bt5 = QPushButton('请输入游戏时间',self)

self.bt5.move(330,260)

#推荐按钮

self.bt=QPushButton('推荐开始',self)

self.bt.move(20,400)

#初始化下拉输入框

self.init\_combobox()

#连接按键与槽

self.bt1.clicked.connect(self.timeDialog)

self.bt2.clicked.connect(self.timeDialog)

self.bt3.clicked.connect(self.timeDialog)

self.bt4.clicked.connect(self.timeDialog)

self.bt5.clicked.connect(self.timeDialog)

#连接推荐

self.bt.clicked.connect(self.recommand)

#初始化下拉输入框

def init\_combobox(self):

#增加选项元素

for i in range(len(gamelist)):

self.combobox1.addItem(gamelist[i])

self.combobox2.addItem(gamelist[i])

self.combobox3.addItem(gamelist[i])

self.combobox4.addItem(gamelist[i])

self.combobox5.addItem(gamelist[i])

self.combobox1.setCurrentIndex(-1)

self.combobox2.setCurrentIndex(-1)

self.combobox3.setCurrentIndex(-1)

self.combobox4.setCurrentIndex(-1)

self.combobox5.setCurrentIndex(-1)

#增加自动补全

self.completer = QCompleter(gamelist)

#补全方式

self.completer.setFilterMode(Qt.MatchStartsWith)

self.completer.setCompletionMode(QCompleter.PopupCompletion)

self.combobox1.setCompleter(self.completer)

self.combobox2.setCompleter(self.completer)

self.combobox3.setCompleter(self.completer)

self.combobox4.setCompleter(self.completer)

self.combobox5.setCompleter(self.completer)

def timeDialog(self):

#获取信号

sender = self.sender()

if sender == self.bt1:

#获取下拉输入框1输入的游戏名

gamename = self.combobox1.currentText()

#通过字典game2idx查询获得的游戏名所对应的序列号

gameid = game2idx.get(gamename)

#没有序列号的情况，可以理解为未输入正确的游戏名，或者输入为空

if gameid == None:

#这种情况下生成一个MessageBox报错

reply = QMessageBox.information(self,'Error','请输入正确的游戏名!', QMessageBox.Close)

else:

#输入正确的情况，将游戏名字、ID，分别记录到一个字典里，方便保存与更改

gamedict[1] = gamename

idxdict[1] = gameid

#弹出一个文本输入框，要求输入对应的游戏时长

text, ok = QInputDialog.getDouble(self, '游戏时间', '请输入游戏时间：', min = 0.1)

#如果输入正确，将时长记录到一个字典中，方便保存与更改

if ok:

timedict[1] = text

elif sender == self.bt2:

gamename = self.combobox2.currentText()

gameid = game2idx.get(gamename)

if gameid == None:

reply = QMessageBox.information(self,'Error','请输入正确的游戏名!', QMessageBox.Close)

else:

gamedict[2] = gamename

idxdict[2] = gameid

text, ok = QInputDialog.getDouble(self, '游戏时间', '请输入游戏时间：', min = 0.1)

if ok:

timedict[2] = text

elif sender == self.bt3:

gamename = self.combobox3.currentText()

gameid = game2idx.get(gamename)

if gameid == None:

reply = QMessageBox.information(self,'Error','请输入正确的游戏名!', QMessageBox.Close)

else:

gamedict[3] = gamename

idxdict[3] = gameid

text, ok = QInputDialog.getDouble(self, '游戏时间', '请输入游戏时间：', min = 0.1)

if ok:

timedict[3] = text

elif sender == self.bt4:

gamename = self.combobox4.currentText()

gameid = game2idx.get(gamename)

if gameid == None:

reply = QMessageBox.information(self,'Error','请输入正确的游戏名!', QMessageBox.Close)

else:

gamedict[4] = gamename

idxdict[4] = gameid

text, ok = QInputDialog.getDouble(self, '游戏时间', '请输入游戏时间：', min = 0.1)

if ok:

timedict[4] = text

elif sender == self.bt5:

gamename = self.combobox5.currentText()

gameid = game2idx.get(gamename)

if gameid == None:

reply = QMessageBox.information(self,'Error','请输入正确的游戏名!', QMessageBox.Close)

else:

gamedict[5] = gamename

idxdict[5] = gameid

text, ok = QInputDialog.getDouble(self, '游戏时间', '请输入游戏时间：', min = 0.1)

if ok:

timedict[5] = text

def recommand(self):

#验证是否存在没有写入的数据

c = 0

for i in range(1,6):

if gamedict[i] == "NULL":

c+=1

if idxdict[i] == "NULL":

c+=1

if timedict[i] == "NULL":

c+=1

#全部写完的情况

if c == 0:

#将字典转化为列表

usertime = list(timedict.values())

useridx = list(idxdict.values())

#调用模型

allrecidx = UserSimilarity(useridx,usertime)

#降序排列数据

rr = np.argsort(-allrecidx)

#获取排行前五的游戏ID

top\_k = rr[:5]

#将ID对应的游戏名字输入数组

for i in top\_k:

recgame.append(idx2game[i])

#将数组转化为字符串并输出

reclist = ','.join(recgame)

reply = QMessageBox.information(self,'推荐的游戏','给您推荐的游戏是'+reclist, QMessageBox.Close)

#存在没有写完的数据，要求重新写入

else:

reply = QMessageBox.information(self,'Error','请输入全部数据!', QMessageBox.Close)

#主函数

if \_\_name\_\_ == "\_\_main\_\_":

app = QApplication(sys.argv)

w = Recommandation()

w.show()

sys.exit(app.exec\_())

项目14 语音识别和字幕推荐系统

14.3 模块实现

14.3.1 数据预处理

(1)下载使用SDK

根据pip工具使用pip install baidu-aip下载

调用代码为：

from aip import AipSpeech

APP\_ID = 'XXXXXXX'

API\_KEY = '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'

SECRET\_KEY = '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'

client = AipSpeech(APP\_ID, API\_KEY, SECRET\_KEY)

#读取文件

def get\_file\_content(file\_path):

with open(file\_path, 'rb') as fp:

return fp.read()

result=client.asr(get\_file\_content('test2.wav'),'wav',16000,{

'dev\_pid': 1537,

})

print(type(result))

print(result)

(2)不需要下载SDK

#合成请求token的URL

url = baidu\_server + "grant\_type=" + grant\_type + "&client\_id=" + client\_id + "&client\_secret=" + client\_secret

#获取token

res = urllib.request.urlopen(url).read()

data = json.loads(res.decode('utf-8'))

token = data["access\_token"]

print(token)

下载配置所需要的库相关操作如下：

(1)安装moviepy库

14.3.2 翻译

class baidu\_Translate():  
 def \_\_init\_\_(self):  
 self.js = js2py.eval\_js('''  
 var i = null;  
 function n(r, o) {  
 for (var t = 0; t < o.length - 2; t += 3) {  
 var a = o.charAt(t + 2);  
 a = a >= "a" ? a.charCodeAt(0) - 87 : Number(a),  
 a = "+" === o.charAt(t + 1) ? r >>> a: r << a,  
 r = "+" === o.charAt(t) ? r + a & 4294967295 : r ^ a  
 }  
 return r  
 }  
 var hash = function e(r,gtk) {  
 var o = r.match(/[\uD800-\uDBFF][\uDC00-\uDFFF]/g);  
 if (null === o) {  
 var t = r.length;  
 t > 30 && (r = "" + r.substr(0, 10) + r.substr(Math.floor(t / 2) - 5, 10) + r.substr( - 10, 10))  
 } else {  
 for (var e = r.split(/[\uD800-\uDBFF][\uDC00-\uDFFF]/), C = 0, h = e.length, f = []; h > C; C++)"" !== e[C] && f.push.a pply(f, a(e[C].split(""))),C !== h - 1 && f.push(o[C]);  
 var g = f.length;  
 g > 30 && (r = f.slice(0, 10).join("")+f.slice(Math.floor(g/ 2) - 5, Math.floor(g / 2) + 5).join("") + f.slice( - 10).join(""))  
 }  
 var u = void 0,  
 u = null !== i ? i: (i = gtk || "") || "";  
 for (var d = u.split("."), m = Number(d[0]) || 0, s = Number(d [1]) || 0, S = [], c = 0, v = 0; v < r.length; v++) {  
 var A = r.charCodeAt(v);  
 128 > A ? S[c++] = A: (2048 > A ? S[c++] = A >> 6 | 192 : (55296 ===(64512 & A) && v + 1 < r.length && 56320 ===(64512&r.charCodeAt(v + 1)) ? (A = 65536 + ((1023 & A) << 10) + (1023 & r.charCodeAt(++v)), S[c++] = A >> 18 | 240, S[c++] = A >> 12 &63 | 128) : S[c++] = A >> 12 | 224,S[c++] = A >> 6 & 63 | 128), S[c++] = 63 & A | 128)  
 }  
 for (  
 var p = m,F = "+-a^+6", D = "+-3^+b+-f", b = 0;  
 b < S.length; b++) p += S[b],p = n(p, F);  
 return p = n(p, D),  
 p ^= s,  
 0 > p && (p = (2147483647 & p) + 2147483648),  
 p %= 1e6,  
 p.toString() + "." + (p ^ m)  
 }  
 ''')  
 headers = {  
 'user-agent': 'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.96 Safari/537.36', }  
 self.session = requests.Session()  
 self.session.get('https://fanyi.baidu.com', headers=headers)  
 response = self.session.get('https://fanyi.baidu.com', headers=headers)  
 self.token = re.findall("token: '(.\*?)',", response.text)[0]  
 self.gtk = '320305.131321201' # re.findall("window.gtk = '(.\*?)';", response.text, re.S)[0]  
 def translate(self, query, from\_lang='en', to\_lang='zh'):  
 #语言检测  
 self.session.post('https://fanyi.baidu.com/langdetect', data={'query': query})  
 #单击事件  
 self.session.get('https://click.fanyi.baidu.com/?src=1&locate=zh&actio n=query&type=1&page=1')  
 #翻译  
 data = {  
 'from': from\_lang,  
 'to': to\_lang,  
 'query': query,  
 'transtype': 'realtime',  
 'simple\_means\_flag': '3',  
 'sign': self.js(query, self.gtk),  
 'token': self.token  
 }  
 response = self.session.post('https://fanyi.baidu.com/v2transapi', data=data)  
 json = response.json()  
 if 'error' in json:  
 pass  
 #return 'error: {}'.format(json['error'])  
 else:  
 return response.json()['trans\_result']['data'][0]['dst']

14.3.3 格式转换

def separate\_audio(file\_path, save\_path):

#视频转音频，音频编码要求：.wav格式、采样率 16000、16bit 位深、单声道

audio\_file = save\_path + '\\tmp.wav'

audio = AudioFileClip(file\_path)

audio.write\_audiofile(audio\_file, ffmpeg\_params=['-ar', '16000', '-ac', '1'], logger=None)

return audio\_file

14.3.4 音频切割

def cut\_point(path, dbfs=1.25): #音频切割函数

sound = AudioSegment.from\_file(path, format="wav")

tstamp\_list = detect\_silence(sound, 600, sound.dBFS \* dbfs, 1)

timelist = []

for i in range(len(tstamp\_list)):

if i == 0:

back = 0

else:

back = tstamp\_list[i - 1][1] / 1000

timelist.append([back, tstamp\_list[i][1] / 1000])

min\_len = 0.5 #切割所得音频不短于0.5s，不长于5s

max\_len = 5

result = []

add = 0

total = len(timelist)

for x in range(total):

if x + add < total:

into, out = timelist[x + add]

if out-into>min\_len and out-into<max\_len and x + add +1 < total:

add += 1

out = timelist[x + add][1]

result.append([into, out])

elif out - into > max\_len:

result.append([into, out])

else:

break

return result

14.3.5 语音识别

class baidu\_SpeechRecognition(): #调用百度语音识别API进行操作

def \_\_init\_\_(self, dev\_pid):

Speech\_APP\_ID = '19712136'

Speech\_API\_KEY = 'Loo4KbNtagchc2BLdCnHEnZl'

Speech\_SECRET\_KEY = 'DO4UlSnw7FzpodU2G3yXQSHLv6Q2inN8'

self.dev\_pid = dev\_pid

self.SpeechClient = AipSpeech(Speech\_APP\_ID, Speech\_API\_KEY, Speech\_SECRET\_KEY)

self.TranslClient = baidu\_Translate()

def load\_audio(self, audio\_file): #读取加载音频文件

self.source = AudioSegment.from\_wav(audio\_file)

def speech\_recognition(self, offset, duration, fanyi):

#语音识别，根据要求的参数进行设置

data = self.source[offset \* 1000:duration \* 1000].raw\_data

result = self.SpeechClient.asr(data, 'wav', 16000, {'dev\_pid': self.dev\_pid, })

fanyi\_text = ''

if fanyi:

try:

fanyi\_text = self.TranslClient.translate(result['result'][0])

#调用translate()函数，将识别文本翻译或直接输出

except:

pass

try:

return [result['result'][0], fanyi\_text] #返回所得文本

except:

#print('错误:',result)

return ['', '']

14.3.6 文本切割

def cut\_text(text, length=38):

#文本切割，即断句，一个画面最多单语言字数不超过38，否则将多出的加入下一画面

newtext = ''

if len(text) > length:

while True:

cutA = text[:length]

cutB = text[length:]

newtext += cutA + '\n'

if len(cutB) < 4:

newtext = cutA + cutB

break

elif len(cutB) > length:

text = cutB

else:

newtext += cutB

break

return newtext

return text

14.3.7 main函数

if \_\_name\_\_ == '\_\_main\_\_':

def StartHandle(timeList, save\_path, srt\_mode=2, result\_print=False):

index = 0

total = len(timeList)

a\_font = r'{\fn微软雅黑\fs14}' #中、英字幕字体设置

b\_font = r'{\fn微软雅黑\fs10}'

fanyi = False if srt\_mode == 1 else True

file\_write = open(save\_path, 'a', encoding='utf-8')

for x in range(total):

into, out = timelist[x]

timeStamp=format\_time(into - 0.2) +'--> '+ format\_time(out- 0.2)

result=baidufanyi.speech\_recognition(into+0.1,out - 0.1, fanyi)

if result\_print:

if srt\_mode == 0:

print(timeStamp, result[0])

else:

print(timeStamp, result)

else:

progressbar(total, x, '识别中...&& - {0}/{1}'.format('%03d' % (total), '%03d' % (x)), 44)

#将切割后所得的识别文本结果按顺序写入，中、英、中英双语不同

if len(result[0]) > 1:

index += 1

text = str(index) + '\n' + timeStamp + '\n'

if srt\_mode == 0: # 仅中文

text += a\_font + cut\_text(result[1])

elif srt\_mode == 1: # 仅英文

text += b\_font + cut\_text(result[0])

else: #中文+英文

text+=a\_font+cut\_text(result[1])+'\n'+b\_font + result[0]

text = text.replace('\u200b', '') + '\n\n'

file\_write.write(text)

file\_write.close()

if not result\_print:

progressbar(total,total,'识别中...&&-{0}/{1}'.format('%03d'% (total), '%03d' % (total)), 44)

os.system('cls')

wav\_path = os.environ.get('TEMP')

#语音模型，1536为普通话+简单英文，1537为普通话，1737为英语，1637为粤语，1837川话，1936普通话远场

pid\_list = 1536, 1537, 1737, 1637, 1837, 1936

#设置参数

print('[ 百度语音识别字幕生成器 - by Teri ]\n')

\_\_line\_\_print\_\_('1 模式选择')

input\_dev\_pid = input('请选择识别模式:\n'

'\n (1)普通话,'

'\n (2)普通话+简单英语,'

'\n (3)英语,'

'\n (4)粤语,'

'\n (5)四川话,'

'\n (6)普通话-远场'

'\n\n请输入一个选项(默认3):')

\_\_line\_\_print\_\_('2 字幕格式')

input\_srt\_mode = input('请选择字幕格式:\n'

'\n (1)中文,'

'\n (2)英文,'

'\n (3)中文+英文，'

'\n\n请输入一个选项(默认3):')

\_\_line\_\_print\_\_('3 实时输出')

input\_print = input('是否实时输出结果到屏幕? (默认:否/y:输出):').upper()

#处理参数，根据用户输入给出相应参数

dev\_pid = int(input\_dev\_pid) if input\_dev\_pid else 3

dev\_pid -= 1

srt\_mode = int(input\_srt\_mode) if input\_srt\_mode else 3

srt\_mode -= 1

re\_print = True if input\_print == 'Y' else False

#输入文件

\_\_line\_\_print\_\_('4 打开文件')

input\_file = input('请拖入一个文件或文件夹并按回车:').strip('"')

video\_file = []

if not os.path.isdir(input\_file):

video\_file = [input\_file]

else:

file\_list = file\_filter(input\_file)

for a, b in file\_list:

video\_file.append(a + '\\' + b)

#执行确认

select\_dev = ['普通话', '普通话+简单英语', '英语', '粤语', '四川话', '普通话-远场']

select\_mode = ['中文', '英文', '中文+英文']

\_\_line\_\_print\_\_('5 确认执行')

input('当前的设置:\n识别模式: {0}, 字幕格式: {1}, 输出结果: {2}\n当前待处理文件 {3} 个\n请按下回车开始处理...'.format(

select\_dev[dev\_pid],

select\_mode[srt\_mode],

'是' if re\_print else '否',

len(video\_file)

))

#批量处理，调用所设函数进行处理工作

total\_file = len(video\_file)

total\_time = time.time()

baidufanyi = baidu\_SpeechRecognition(pid\_list[dev\_pid])

for i in range(total\_file): #在所给文件范围内循环运行

item\_time = time.time() #项目时间

file\_name = video\_file[i].split('\\')[-1]

print('\n>>>>>>>> ...正在处理音频... <<<<<<<<', end='')

audio\_file = separate\_audio(video\_file[i], wav\_path) #视频转音频

timelist = cut\_point(audio\_file, dbfs=1.15) #音频切割

if timelist:

print('\r>>>>>>>> 当前:{} 预计:{} <<<<<<<<'.format(

'%03d' % (i),

countTime(len(timelist) \* 5, now=False)

))

srt\_name = video\_file[i][:video\_file[i].rfind('.')] + '.srt'

#根据时间将输出循环写入字幕文件

baidufanyi.load\_audio(audio\_file)

StartHandle(timelist, srt\_name, srt\_mode, re\_print)

print('\n{} 处理完成, 本次用时{}'.format(file\_name, countTime(item\_time)))

else:

print('音频参数错误')

#执行完成，统计所用时间

input('全部完成, 处理了{}个文件, 全部用时{}'.format(total\_file, countTime(total\_time)))

#本部分包括活动类、模块的相关函数、主函数代码

from moviepy.editor import AudioFileClip

from pydub import AudioSegment

from pydub.silence import detect\_silence

from aip import AipSpeech

import os

import time

import re

import requests

import js2py

class baidu\_Translate():

def \_\_init\_\_(self):

self.js = js2py.eval\_js('''

var i = null;

function n(r, o) {

for (var t = 0; t < o.length - 2; t += 3) {

var a = o.charAt(t + 2);

a = a >= "a" ? a.charCodeAt(0) - 87 : Number(a),

a = "+" === o.charAt(t + 1) ? r >>> a: r << a,

r = "+" === o.charAt(t) ? r + a & 4294967295 : r ^ a

}

return r

}

var hash = function e(r,gtk) {

var o = r.match(/[\uD800-\uDBFF][\uDC00-\uDFFF]/g);

if (null === o) {

var t = r.length;

t > 30 && (r = "" + r.substr(0, 10) + r.substr(Math.floor(t / 2) - 5, 10) + r.substr( - 10, 10))

} else {

for (var e = r.split(/[\uD800-\uDBFF][\uDC00-\uDFFF]/), C = 0, h = e.length, f = []; h > C; C++)"" !== e[C] && f.push.apply(f, a(e[C].split(""))),

C !== h - 1 && f.push(o[C]);

var g = f.length;

g > 30 && (r = f.slice(0, 10).join("") + f.slice(Math.floor(g / 2) - 5, Math.floor(g / 2) + 5).join("") + f.slice( - 10).join(""))

}

var u = void 0,

u = null !== i ? i: (i = gtk || "") || "";

for (var d = u.split("."), m = Number(d[0]) || 0, s = Number(d[1]) || 0, S = [], c = 0, v = 0; v < r.length; v++) {

var A = r.charCodeAt(v);

128 > A ? S[c++] = A: (2048 > A ? S[c++] = A >> 6 | 192 : (55296 === (64512 & A) && v + 1 < r.length && 56320 === (64512 & r.charCodeAt(v + 1)) ? (A = 65536 + ((1023 & A) << 10) + (1023 & r.charCodeAt(++v)), S[c++] = A >> 18 | 240, S[c++] = A >> 12 & 63 | 128) : S[c++] = A >> 12 | 224, S[c++] = A >> 6 & 63 | 128), S[c++] = 63 & A | 128)

}

for (

var p = m,F = "+-a^+6", D = "+-3^+b+-f", b = 0;

b < S.length; b++) p += S[b],p = n(p, F);

return p = n(p, D),

p ^= s,

0 > p && (p = (2147483647 & p) + 2147483648),

p %= 1e6,

p.toString() + "." + (p ^ m)

}

''')

headers = {

'user-agent': 'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.96 Safari/537.36', }

self.session = requests.Session()

self.session.get('https://fanyi.baidu.com', headers=headers)

response = self.session.get('https://fanyi.baidu.com', headers=headers)

self.token = re.findall("token: '(.\*?)',", response.text)[0]

self.gtk = '320305.131321201' # re.findall("window.gtk = '(.\*?)';", response.text, re.S)[0]

def translate(self, query, from\_lang='en', to\_lang='zh'):

#语言检测

self.session.post('https://fanyi.baidu.com/langdetect', data={'query': query})

#单击事件 self.session.get('https://click.fanyi.baidu.com/?src=1&locate=zh&action=query&type=1&page=1')

#翻译

data = {

'from': from\_lang,

'to': to\_lang,

'query': query,

'transtype': 'realtime',

'simple\_means\_flag': '3',

'sign': self.js(query, self.gtk),

'token': self.token

}

response =self.session.post('https://fanyi.baidu.com/v2transapi', data=data)

json = response.json()

if 'error' in json:

pass

#return 'error: {}'.format(json['error'])

else:

return response.json()['trans\_result']['data'][0]['dst']

class baidu\_SpeechRecognition():

def \_\_init\_\_(self, dev\_pid):

#百度语音识别API

Speech\_APP\_ID = '19712136'

Speech\_API\_KEY = 'Loo4KbNtagchc2BLdCnHEnZl'

Speech\_SECRET\_KEY = 'DO4UlSnw7FzpodU2G3yXQSHLv6Q2inN8'

self.dev\_pid = dev\_pid

self.SpeechClient = AipSpeech(Speech\_APP\_ID, Speech\_API\_KEY, Speech\_SECRET\_KEY)

self.TranslClient = baidu\_Translate()

def load\_audio(self, audio\_file):

self.source = AudioSegment.from\_wav(audio\_file)

def speech\_recognition(self, offset, duration, fanyi):

data = self.source[offset \* 1000:duration \* 1000].raw\_data

result = self.SpeechClient.asr(data, 'wav', 16000, {'dev\_pid': self.dev\_pid, })

fanyi\_text = ''

if fanyi:

try:

fanyi\_text = self.TranslClient.translate(result['result'][0])

except:

pass

try:

return [result['result'][0], fanyi\_text]

except:

#print('错误:',result)

return ['', '']

def cut\_point(path, dbfs=1.25):

sound = AudioSegment.from\_file(path, format="wav")

tstamp\_list = detect\_silence(sound, 600, sound.dBFS \* dbfs, 1)

timelist = []

for i in range(len(tstamp\_list)):

if i == 0:

back = 0

else:

back = tstamp\_list[i - 1][1] / 1000

timelist.append([back, tstamp\_list[i][1] / 1000])

min\_len = 0.5

max\_len = 5

result = []

add = 0

total = len(timelist)

for x in range(total):

if x + add < total:

into, out = timelist[x + add]

if out-into>min\_len and out - into < max\_len and x + add + 1 < total:

add += 1

out = timelist[x + add][1]

result.append([into, out])

elif out - into > max\_len:

result.append([into, out])

else:

break

return result

def cut\_text(text, length=38):

newtext = ''

if len(text) > length:

while True:

cutA = text[:length]

cutB = text[length:]

newtext += cutA + '\n'

if len(cutB) < 4:

newtext = cutA + cutB

break

elif len(cutB) > length:

text = cutB

else:

newtext += cutB

break

return newtext

return text

def progressbar(total, temp, text='&&', lenght=40): #定义进度栏

content = '\r' + text.strip().replace('&&', '[{0}{1}]{2}%')

percentage = round(temp / total \* 100, 2)

a = round(temp / total \* lenght)

b = lenght - a

print(content.format('■' \* a, '□' \* b, percentage), end='')

def format\_time(seconds): #定义时间格式

sec = int(seconds)

m, s = divmod(sec, 60)

h, m = divmod(m, 60)

fm = int(str(round(seconds, 3)).split('.')[-1])

return "%02d:%02d:%02d,%03d" % (h, m, s, fm)

def separate\_audio(file\_path, save\_path): #定义音频间隔

audio\_file = save\_path + '\\tmp.wav'

audio = AudioFileClip(file\_path)

audio.write\_audiofile(audio\_file, ffmpeg\_params=['-ar', '16000', '-ac', '1'], logger=None)

return audio\_file

def file\_filter(path, alldir=False): #定义文件过滤

key = ['mp4', 'mov']

if alldir:

dic\_list = os.walk(path)

else:

dic\_list = os.listdir(path)

find\_list = []

for i in dic\_list:

if os.path.isdir(i[0]):

header = i[0]

file = i[2]

for f in file:

for k in key:

if f.rfind(k) != -1:

find\_list.append([header, f])

else:

for k in key:

if i.rfind(k) != -1:

find\_list.append([path, i])

if find\_list:

find\_list.sort(key=lambda txt: re.findall(r'\d+', txt[1])[0])

return find\_list

def countTime(s\_time, now=True): #定义累计时间

if now: s\_time = (time.time() - s\_time)

m, s = divmod(int(s\_time), 60)

return '{}分{}秒'.format('%02d' % (m), '%02d' % (s))

def \_\_line\_\_print\_\_(txt='-' \* 10): #定义打印

print('\n' + '-' \* 10 + ' ' + txt + ' ' + '-' \* 10 + '\n')

if \_\_name\_\_ == '\_\_main\_\_': #主函数

def StartHandle(timeList, save\_path, srt\_mode=2, result\_print=False):

index = 0

total = len(timeList)

a\_font = r'{\fn微软雅黑\fs14}'

b\_font = r'{\fn微软雅黑\fs10}'

fanyi = False if srt\_mode == 1 else True

file\_write = open(save\_path, 'a', encoding='utf-8')

for x in range(total):

into, out = timelist[x]

timeStamp = format\_time(into - 0.2) + ' --> ' + format\_time(out - 0.2)

result = baidufanyi.speech\_recognition(into + 0.1, out - 0.1, fanyi)

if result\_print:

if srt\_mode == 0:

print(timeStamp, result[0])

else:

print(timeStamp, result)

else:

progressbar(total, x, '识别中...&& - {0}/{1}'.format('%03d' % (total), '%03d' % (x)), 44)

if len(result[0]) > 1:

index += 1

text = str(index) + '\n' + timeStamp + '\n'

if srt\_mode == 0: #仅中文

text += a\_font + cut\_text(result[1])

elif srt\_mode == 1: #仅英文

text += b\_font + cut\_text(result[0])

else: #中文+英文

text += a\_font + cut\_text(result[1]) + '\n' + b\_font + result[0]

text = text.replace('\u200b', '') + '\n\n'

file\_write.write(text)

file\_write.close()

if not result\_print:

progressbar(total, total, '识别中...&& - {0}/{1}'.format('%03d' % (total), '%03d' % (total)), 44)

os.system('cls')

wav\_path = os.environ.get('TEMP')

#语音模型

pid\_list = 1536, 1537, 1737, 1637, 1837, 1936

#设置参数

print('[ 百度语音识别字幕生成器 - by 谷健&任家旺 ]\n')

\_\_line\_\_print\_\_('1 模式选择')

input\_dev\_pid = input('请选择识别模式:\n'

'\n (1)普通话,'

'\n (2)普通话+简单英语,'

'\n (3)英语,'

'\n (4)粤语,'

'\n (5)四川话,'

'\n (6)普通话-远场'

'\n\n请输入一个选项(默认3):')

\_\_line\_\_print\_\_('2 字幕格式')

input\_srt\_mode = input('请选择字幕格式:\n'

'\n (1)中文,'

'\n (2)英文,'

'\n (3)中文+英文，'

'\n\n请输入一个选项(默认3):')

\_\_line\_\_print\_\_('3 实时输出')

input\_print = input('是否实时输出结果到屏幕? (默认:否/y:输出):').upper()

#处理参数

dev\_pid = int(input\_dev\_pid) if input\_dev\_pid else 3

dev\_pid -= 1

srt\_mode = int(input\_srt\_mode) if input\_srt\_mode else 3

srt\_mode -= 1

re\_print = True if input\_print == 'Y' else False

#输入文件

\_\_line\_\_print\_\_('4 打开文件')

input\_file = input('请拖入一个文件或文件夹并按回车:').strip('"')

video\_file = []

if not os.path.isdir(input\_file):

video\_file = [input\_file]

else:

file\_list = file\_filter(input\_file)

for a, b in file\_list:

video\_file.append(a + '\\' + b)

#执行确认

select\_dev = ['普通话', '普通话+简单英语', '英语', '粤语', '四川话', '普通话-远场']

select\_mode = ['中文', '英文', '中文+英文']

\_\_line\_\_print\_\_('5 确认执行')

input('当前的设置:\n识别模式: {0}, 字幕格式: {1}, 输出结果: {2}\n当前待处理文件 {3} 个\n请按下回车开始处理...'.format(

select\_dev[dev\_pid],

select\_mode[srt\_mode],

'是' if re\_print else '否',

len(video\_file)

))

#批量处理

total\_file = len(video\_file)

total\_time = time.time()

baidufanyi = baidu\_SpeechRecognition(pid\_list[dev\_pid])

for i in range(total\_file):

item\_time = time.time()

file\_name = video\_file[i].split('\\')[-1]

print('\n>>>>>>>> ...正在处理音频... <<<<<<<<', end='')

audio\_file = separate\_audio(video\_file[i], wav\_path)

timelist = cut\_point(audio\_file, dbfs=1.15)

if timelist:

print('\r>>>>>>>> 当前:{} 预计:{} <<<<<<<<'.format(

'%03d' % (i),

countTime(len(timelist) \* 5, now=False)

))

srt\_name = video\_file[i][:video\_file[i].rfind('.')] + '.srt'

baidufanyi.load\_audio(audio\_file)

StartHandle(timelist, srt\_name, srt\_mode, re\_print)

print('\n{} 处理完成, 本次用时{}'.format(file\_name, countTime(item\_time)))

else:

print('音频参数错误')

#执行完成

input('全部完成, 处理了{}个文件, 全部用时{}'.format(total\_file, countTime(total\_time)))

项目15 发型推荐系统设计

15.3 模块实现

15.3.1 Face++·API调用

相关代码如下：

#导入相应库文件

import requests

from json import JSONDecoder  
import urllib.error  
import base64  
import os  
import time

4)性别检测函数

相关代码如下：

#用户性别，参数为图片路径

def detect\_gender(filepath):

#URL  
 http\_url1 = 'https://api-cn.faceplusplus.com/facepp/v3/detect'

#账号密码  
 key = "CNsZZXKA4M2qzlz8lKw5ML0BRSitwHfW"  
 secret = "c8udJa\_mDz\_KIRAhrvxV9w5PbrrVtTM0"

#数据提交  
 data1= {'api\_key': key, 'api\_secret': secret, 'return\_attributes': "gender"}  
 files1 = {"image\_file": open(filepath, "rb")}

#数据获取  
 response1 = requests.post(http\_url1, data=data1, files=files1)

#数据进行utf-8编译后解码处理  
 req\_con1 = response1.content.decode('utf-8')  
 req\_dict1 = JSONDecoder().decode(req\_con1)

#网络状态

#判断获取到的数据是否为正确数据  
 if response1.status\_code == requests.codes.ok:

#提取正确数据的性别相关内容  
 sex = req\_dict1["faces"][0]['attributes']['gender']['value']  
 print(req\_dict1["faces"][0]['attributes']['gender']['value'])  
 return sex  
 else:

#数据错误则打印性别识别失败，返回空值  
 print('faile to detect\_gender')  
 return None

5)脸型检测函数

相关代码如下：

#用户脸型  
def detect\_face\_type(filepath):  
 http\_url2 = 'https://api-cn.faceplusplus.com/facepp/v1/facialfeatures'  
 key = "CNsZZXKA4M2qzlz8lKw5ML0BRSitwHfW"  
 secret = "c8udJa\_mDz\_KIRAhrvxV9w5PbrrVtTM0"  
 data2 = {'api\_key': key, 'api\_secret': secret}  
 files2 = {"image\_file": open(filepath, "rb")}  
 response2 = requests.post(http\_url2, data=data2, files=files2)  
 req\_con2 = response2.content.decode('utf-8')  
 req\_dict2 = JSONDecoder().decode(req\_con2)  
 if response2.status\_code == requests.codes.ok:

#提取正确数据的脸型相关内容  
 face\_type = req\_dict2["result"]["face"]["face\_type"]  
 print(req\_dict2["result"]["face"]["face\_type"])  
 return face\_type  
 else:

#数据错误则打印脸型识别失败，返回空值  
 print('faile to detect\_face\_type')  
 return None

6)人脸融合主函数

相关代码如下：

#识别人脸关键点信息

def find\_face(imgpath):  
 http\_url1 = 'https://api-cn.faceplusplus.com/facepp/v3/detect'  
 key = "CNsZZXKA4M2qzlz8lKw5ML0BRSitwHfW"  
 secret = "c8udJa\_mDz\_KIRAhrvxV9w5PbrrVtTM0"

#"return\_landmark": 2表示获取106个人脸关键点信息  
 data1 = {'api\_key': key, 'api\_secret': secret, "return\_landmark": 2}

#判断路径图片是否存在  
 if os.path.isfile(imgpath)==False:  
 return None  
 files = {"image\_file": open(imgpath, "rb")}  
 response1 = requests.post(http\_url1, data=data1, files=files)  
 req\_con1 = response1.content.decode('utf-8')  
 req\_dict1 = JSONDecoder().decode(req\_con1)  
 if response1.status\_code == requests.codes.ok:

#获取人脸关键点信息  
 face\_rectangle = req\_dict1["faces"][0]['face\_rectangle']  
 return face\_rectangle  
 else:  
 print('faile to find\_face')  
 #number表示换脸的相似度  
 #将上述关键点信息调用，实现人脸融合

#参数为用户人像路径、模型发型路径、生成效果图路径

#number表示人脸融合相似度范围为0~100  
def merge\_face(image\_url\_1, image\_url\_2, image\_url, number):  
 ff1 = find\_face(image\_url\_1)  
 ff2 = find\_face(image\_url\_2)  
 if ff1 and ff2:  
 rectangle1 = str(str(ff1['top']) + "," + str(ff1['left']) + "," + str(ff1['width']) + "," + str(ff1['height']))  
 rectangle2 = str(ff2['top']) + "," + str(ff2['left']) + "," + str(ff2['width']) + "," + str(ff2['height'])  
 url\_add = "https://api-cn.faceplusplus.com/imagepp/v1/mergeface"  
 f1 = open(image\_url\_1, 'rb')  
 f1\_64 = base64.b64encode(f1.read())  
 f1.close()  
 f2 = open(image\_url\_2, 'rb')  
 f2\_64 = base64.b64encode(f2.read())  
 f2.close()  
 data = {"api\_key": "CNsZZXKA4M2qzlz8lKw5ML0BRSitwHfW", "api\_secret": "c8udJa\_mDz\_KIRAhrvxV9w5PbrrVtTM0",  
 "template\_base64": f1\_64, "template\_rectangle": rectangle1,  
 "merge\_base64": f2\_64, "merge\_rectangle": rectangle2, "merge\_rate": number}  
 response = requests.post(url\_add, data=data)  
 req\_con = response.content.decode('utf-8')  
 req\_dict = JSONDecoder().decode(req\_con)  
 #判断网络状态  
 if response.status\_code == requests.codes.ok:  
 result = req\_dict['result']  
 imgdata = base64.b64decode(result)  
 file = open(image\_url, 'wb')

#图片保存到相应路径  
 file.write(imgdata)  
 file.close()  
 else:  
 print('faile to merge\_face')  
 return None

15.3.2 数据爬取

爬虫实现

本部分包括引入库文件、爬虫初始化。

#引入库文件

from selenium import webdriver  
import urllib.request  
import re  
import requests  
from json import JSONDecoder  
import urllib.error  
import base64  
import time  
from selenium.webdriver.chrome.options import Options

#爬虫初始化，浏览器初始化并模拟滚动条向下滚动

word ="男生发型"  
#创建一个参数对象，用来控制chrome以无界面模式打开  
chrome\_options = Options()  
chrome\_options.add\_argument('--headless')  
chrome\_options.add\_argument('--disable-gpu')  
#驱动路径  
path = r'C:\Users\ZBLi\Desktop\1801\day05\ziliao\chromedriver.exe'  
#创建浏览器对象  
browser = webdriver.Chrome(executable\_path="C:\Program Files (x86)\Google\chromedriver.exe", chrome\_options=chrome\_options)  
#参数添加  
browser.maximize\_window() #最大化

#地址加入关键词实现网址获取  
browser.get('http://image.baidu.com/search/index?tn=baiduimage&ps=1&ct=201326592&lm=-1&cl=2&nc=1&ie=utf-8&word='+word)  
js = 'var action=document.documentElement.scrollTop=10001'  
#设置滚动条距离顶部的位置为 10000， 超过10000就是最底部  
for i in range(5):#共执行5次脚本 实现滑轮向下滚动5次  
 browser.execute\_script(js) #执行脚本  
 time.sleep(2) #休眠2s

#读取源代码  
data=browser.page\_source  
#爬虫  
k = re.split(r'\s+',data)  
s = []  
sp = []

#进行正则表达式匹配  
for i in k :  
 if re.match(r'data-objurl=',i) :  
 if re.match(r'.\*?jpg"', i)or re.match(r'.\*?png"', i):  
 s.append(i)

for it in s :  
 if (re.match(r'.\*?png"',it) or re.match(r'.\*?jpg"',it) ):  
 sp.append(it)

#将匹配到的多余部分进行删减精准定位URL信息  
for it in sp:  
 m = re.search(r'data-objurl="(.\*?)"',it)  
 iturl =m.group(1)  
 #url  
 print(iturl)  
 itdata = None

#避免出现数据读取过慢而导致超时问题

#如果规定时间内无法识别则跳过并睡眠0.1s  
 try:  
 itdata = urllib.request.urlopen(iturl,data=None,timeout=1).read()  
 except:  
 time.sleep(0.1)

#itdata为爬取图片的URL  
 if itdata==None:  
 continue

图片分类存储，采用Face++ 爬取图片的性别与脸型信息并存储(存储图片类型共分为14种：两种性别、七种脸型)，以瓜子脸为例 ，相关代码如下：

#判断脸型是否存在并将其导入变量

if (detect\_face\_type(itdata) and detect\_gender(itdata)):  
 sex = detect\_gender(itdata)  
 face\_type = detect\_face\_type(itdata)  
 #瓜子脸

#判断脸型与性别，符合则存储至相应位置并重命名为Male\_ pointed\_faceNUM   
 if sex == 'Male' and face\_type == 'pointed\_face':

#存储到指定区域并分类别命名  
 f = open('E:\BeautifulPicture\\' + sex + '\_' + face\_type + str(Male\_pointed\_num) + '.jpg', "wb")

#序号逐渐增加代表每一种脸型的总数不断增加  
 Male\_pointed\_num += 1  
 f.write(itdata)  
 f.close()  
 if sex == 'Female' and face\_type == 'pointed\_face':  
 f = open('E:\BeautifulPicture\\' + sex + '\_' + face\_type + str(Female\_pointed\_num) + '.jpg', "wb")  
 Female\_pointed\_num += 1  
 f.write(itdata)  
 f.close()

#最后退出终止浏览器

browser.execute\_script(js)  
browser.quit()

15.3.3 模型构建

1.库函数调用

相关代码如下：

#调用相应库实现功能

#-\*- coding: utf-8 -\*-  
import requests  
from json import JSONDecoder  
import urllib.error  
import base64  
import os  
import time

#模拟用户面部图片并设定路径，后面GUI调试将变为可视化打开图片形式

filepath = r"E:/new/Female\_oval\_face9.jpg"  
sex = detect\_gender(filepath)  
face\_type = detect\_face\_type(filepath)

#人脸融合-核心函数core()

#以瓜子脸为例，假设模板最多25种发型推荐

number = 25

Male\_pointed\_num = 1

Female\_pointed\_num = 1

if sex == 'Male' and face\_type == 'pointed\_face':

#i从1~25进行循环  
 for i in range(number):

#判断是否存在该图片路径，存在则进行人脸融合  
 if os.path.isfile(r"E:\app\picture\Male\_pointed\_face" + str(Male\_pointed\_num) + ".jpg"):  
 exm=r"E:\app\picture\Male\_pointed\_face" + str(Male\_pointed\_num)+".jpg"  
 result = r"E:\app\picture1\\" + str(Male\_pointed\_num) + ".jpg"

#人脸融合更加真实，设定相似度为90/100  
 merge\_face(exm, filepath, result, 90)

#成功则进行下一部分循环  
 Male\_pointed\_num += 1  
 print("人脸融合成功")

#女性瓜子脸效果同上  
#Female\_pointed\_face(瓜子脸)  
if sex == 'Female' and face\_type == 'pointed\_face':  
 for i in range(number):

#每次循环计数加一判断发型模板是否存在，存在则进行人脸融合  
 if os.path.isfile(r"E:\app\picture\Female\_pointed\_face" + str(Female\_pointed\_num) + ".jpg"):  
 exm = r"E:\app\picture\Female\_pointed\_face" + str(Female\_pointed\_num) + ".jpg"  
 result = r"E:\app\picture1\\" + str(Female\_pointed\_num) + ".jpg"  
 merge\_face(exm, filepath, result, 90)  
 Female\_pointed\_num += 1  
 print("人脸融合成功")

15.3.4 用户界面设计

相关代码如下：

1)需要调用的库文件

相关代码如下：

#-\*- coding:utf-8 -\*-  
from tkinter import \*  
from PIL import Image,ImageTk  
from tkinter.filedialog import askopenfilename  
import requests  
from json import JSONDecoder  
import urllib.error  
import base64  
import os

2)读取用户人脸图片位置

相关代码如下：

#获取可视化打开的文件路径

def getpathfile():  
 root = Tk()

#GUI界面图标设置

root.iconbitmap(r'e:\app\ling.ico')  
 width = 500  
 height = 500

#界面居中显示  
 screenwidth = root.winfo\_screenwidth()  
 screenheight = root.winfo\_screenheight()  
 alignstr = '%dx%d+%d+%d' % (width, height, (screenwidth - width) / 2, (screenheight - height) / 2)  
 root.geometry(alignstr)  
 def choosepic():  
 global filepath  
 path\_ = askopenfilename()  
 path.set(path\_)  
 img\_open=Image.open(file\_entry.get()).resize((350,400),Image.ANTIALIAS)  
 img = ImageTk.PhotoImage(img\_open)  
 image\_label.config(image=img)  
 image\_label.image = img   
 filepath = path\_  
 path = StringVar()  
 Button(root, text='选择图片',font=('Arial', 15),bg="yellow",command=choosepic).pack()  
 Button(root, text='ok',font=('Arial', 15), command=root.destroy).pack()  
 file\_entry = Entry(root, state='readonly', text=path)  
 image\_label = Label(root)  
 image\_label.pack()  
 root.mainloop()

3)判断用户人脸图片是否能成功识别函数

相关代码如下：

#用户人脸图片识别成功显示GUI界面

def yes():  
 root = Tk()

#GUI界面图标设置

root.iconbitmap(r'e:\app\ling.ico')  
 width = 500  
 height = 100

#界面居中显示  
 screenwidth = root.winfo\_screenwidth()  
 screenheight = root.winfo\_screenheight()  
 alignstr = '%dx%d+%d+%d' % (width, height, (screenwidth - width) / 2, (screenheight - height) / 2)  
 root.geometry(alignstr)  
 root.title('识别成功')  
 #text = StringVar()  
 #text.set("已为您搜索"+str(num)+"种合适的发型。。。")  
 Label(root, text='识别成功，请稍后', font=('Arial', 20)).pack()  
 Button(root, text='继续', command=root.destroy).pack()  
 #Label(root, textvariable=text, font=('Arial', 20)).pack()  
 root.mainloop()

#用户人脸图片识别失败显示GUI界面  
def no():  
 root = Tk()

#GUI界面图标设置

root.iconbitmap(r'e:\app\ling.ico')  
 width = 500  
 height = 100  
 screenwidth = root.winfo\_screenwidth()  
 screenheight = root.winfo\_screenheight()  
 alignstr = '%dx%d+%d+%d' % (width, height, (screenwidth - width) / 2, (screenheight - height) / 2)  
 root.geometry(alignstr)  
 root.title('识别失败')  
 Label(root, text='识别失败，建议您重新选择合适图片！',font=('Arial', 20)).pack()  
 Button(root, text='结束', command=root.destroy).pack()  
 #Button(root, text='返回上一步', command=getpathfile).pack()  
 frm = Frame(root).pack()  
 root.mainloop()

4)最终效果图展示函数

相关代码如下：

def show(sex,face\_type):  
 root1 = Tk()

#GUI界面图标设置

root1.iconbitmap(r'e:\app\ling.ico')  
 #设置窗口居中  
 width = 500  
 height = 500

#界面居中显示  
 screenwidth = root1.winfo\_screenwidth()  
 screenheight = root1.winfo\_screenheight()  
 alignstr = '%dx%d+%d+%d' % (width, height, (screenwidth - width) / 2, (screenheight - height) / 2)  
 root1.geometry(alignstr)  
 root1.title('效果展示')

#实现动态参数替换

#即每次改变图片，编号实时更新  
 text = StringVar()  
 image\_label = Label(root1)  
 #默认展示第一张图片  
 pilImage\_pre1 = Image.open(r"E:\app\picture1\\" + str(cou) + ".jpg")  
 pilImage1 = pilImage\_pre1.resize((350, 400), Image.ANTIALIAS)  
 tkImage1 = ImageTk.PhotoImage(image=pilImage1)  
 image\_label.config(image=tkImage1)  
 image\_label.image = tkImage1   
 root1.update\_idletasks() #更新图片，必须update

text.set(str(cou) + "/" + str(num))#更新图片编号

#下一张功能实现  
 def change\_next():  
 global cou

#cou为当前展示图片编号

#num为用户搜索到的所有图片数量  
 if cou<num:  
 cou = cou + 1  
 pilImage\_pre = Image.open(r"E:\app\picture1\\" + str(cou) + ".jpg")  
 pilImage = pilImage\_pre.resize((350, 400), Image.ANTIALIAS)  
 tkImage = ImageTk.PhotoImage(image=pilImage)  
 image\_label.config(image=tkImage)  
 image\_label.image = tkImage #保持参考点  
 root1.update\_idletasks() #更新图片，必须update  
 text.set(str(cou)+"/"+str(num)) #更新图片编号

else:

#达到下限  
 print("error")

#上一张功能实现

def change\_prior():  
 global cou  
 if cou > 1:  
 cou = cou - 1  
 pilImage\_pre = Image.open(r"E:\app\picture1\\" + str(cou) + ".jpg")  
 pilImage = pilImage\_pre.resize((350, 400), Image.ANTIALIAS)  
 tkImage = ImageTk.PhotoImage(image=pilImage)  
 image\_label.config(image=tkImage)  
 image\_label.image = tkImage #保持参考点  
 root1.update\_idletasks() #更新图片，必须update  
 text.set(str(cou)+"/"+str(num)) #更新图片编号  
 else: #达到上限  
 print("error")  
 Button(root1, text='下一张', command=change\_next).pack(side=RIGHT)  
 Button(root1, text='上一张', command=change\_prior).pack(side=LEFT)  
 #Label(root1, text="您的脸型为"+face\_type, font=('Arial', 20)).pack(side=TOP)  
 if face\_type=="long\_face":  
 Label(root1, text="您的脸型为长脸" , font=('Arial', 20)).pack(side=TOP)  
 if sex == "Female":  
 Label(root1, text="长脸的你适合俏皮花苞头、蝴蝶结的发箍配上梨花头", font=('Arial', 15)).pack(side=TOP)  
 if sex == "Male":  
 Label(root1, text="长脸的你适合中分刘海的顺直短发发型", font=('Arial', 15)).pack(side=TOP)

Label(root1, text="生产发型已为您自动保存到E\APP\picture1 文件夹", font=('Arial', 10)).pack(side=BOTTOM)  
 Label(root1, textvariable=text, font=('Arial', 20)).pack(side=BOTTOM)  
 Frame(root1).pack()  
 image\_label.pack()  
 root1.mainloop()

5)模块拼接

def main():

#得到图片路径  
 getpathfile()  
 print(filepath)

#得到用户性别与脸型  
 sex = detect\_gender(filepath)  
 print(sex)  
 face\_type = detect\_face\_type(filepath)  
 print(face\_type)

#判断是否可以进行发型推荐  
 if (sex and face\_type):

#继续  
 yes()  
 print("yes")  
 else:

#失败提示  
 no()  
 print("no")

#传递脸型与性别进行人脸融合  
 core(sex,face\_type)

#num变量作为全局参数记录所有推荐发型总和   
 print("共" + str(num) + "张图片可供选择")

#推荐发型数量大于0则进行展示  
 if num > 0:  
 show(sex, face\_type)  
 else:  
 print("查找失败")

项目16 基于百度AI的垃圾分类推荐系统

16.3 模块实现

16.3.1 PC端垃圾分类

1.获取access\_token

相关代码如下：

import requests

#client\_id 为官网获取的AK，client\_secret 为官网获取的SK

host='https://aip.baidubce.com/oauth/2.0/token?grant\_type=client\_credentials&client\_id=[AK]&client\_secret=[SK]'

response = requests.get(host)

if response:

print(response.json())

2.载入图片函数

垃圾分类图像识别的图片来自本地文件，而百度AI要求识别的图片用64位编码，所以导入win32ui和base64库。Win32ui库用于生成文件对话框，base64库用于对图片文件进行编码。

defimage\_load(): #导入图片函数  
dlg= win32ui.CreateFileDialog(1)

#获取一个PyCFileDialog类的对象，通俗讲就是一个对话框，参数1表示“打开文件”对话框  
dlg.SetOFNInitialDir(os.path.abspath(os.curdir))

#指示对话框打开目录  
dlg.DoModal()

#显示对话框，返回一个整数，这个整数指定对话框的操作  
image\_path = dlg.GetPathName()

#以字符串的形式返回完整的文件名  
f = open(image\_path, **'rb'**)

#打开一个文件，设置需要打开的选项  
image = base64.b64encode(f.read())

#二进制方式打开图片文件  
f.close

#关闭文件  
return image

3.百度AI调用函数

defbaiduai\_query(image):  
request\_url= "https://aip.baidubce.com/rest/2.0/image-classify/v2/advanced\_general"  
access\_token= '24.fdaa8f2c2686c85f8b2eca9f32c1289a.2592000.1588922347.282335-18675408'  
request\_url = request\_url+ "?access\_token=" + access\_token  
 data = parse.urlencode({"image": image})  
headers = {'Content-Type': 'application/x-www-form-urlencoded'}  
request = requests.post(request\_url, data=data, headers=headers)  
r = json.loads(request.text)  
key1 = r['result'][0]['keyword']  
root1 = r['result'][0]['root']   
keyword = [key1,root1]  
return keyword

4.get\_html函数

在网站<https://lajifenleiapp.com/>中获取其网页源码。

defget\_html(url):  
headers = {  
'User-Agent':'Mozilla/5.0(Macintosh; Intel Mac OS X 10\_11\_4)\  
AppleWebKit/537.36(KHTML, like Gecko) Chrome/52 .0.2743. 116 Safari/537.36'  
} #模拟浏览器访问  
response = requests.get(url,headers= headers) #请求访问网站  
html = response.text #获取网页源码  
return html #返回网页源码

5.main函数

soup= BeautifulSoup(get\_html("https://lajifenleiapp.com/sk/"+query\_word[0]), 'lxml')

#初始化BeautifulSoup库,并设置解析器  
nonex= soup.find('div', attrs={'class': 'col-md-9 col-xs-12'})  
if nonexis None: #节点不存在，说明有该垃圾的相关信息  
laji= soup.find('span', attrs={'style': 'color:#D42121;'})

#该节点是查询的垃圾名字  
shuyu= soup.find('span', attrs={'style': 'color:#FBbC28;'})  
fenlei= soup.find('span', attrs={'style': '#2e2a2b'})

#该节点是垃圾分类  
print(laji.string) #string提取文字  
print(shuyu.string.strip()) #string.strip()可以删除括号  
print(fenlei.string)  
else: #节点存在，说明没有该垃圾的相关信息  
print("没有与此物品名称匹配的词条")  
print("该物品属于：")  
print(query\_word[1])

#query\_word[1]是百度AI调用函数中的root1，即上级分类  
print("请查找与之分类相关的名称。")

2)根据输入的关键字进行垃圾分类

rubbishname= input(**"**输入要查询的垃圾名称：")  
soup = BeautifulSoup(get\_html("https://lajifenleiapp.com/sk/"+rubbishname), 'lxml') #初始化BeautifulSoup库,并设置解析器  
nonex= soup.find('div', attrs={'class': 'col-md-9 col-xs-12'})  
print("正在查询……")  
if nonexis None:  
laji= soup.find('span', attrs={'style': 'color:#D42121;'})  
shuyu= soup.find('span', attrs={'style': 'color:#FBbC28;'})  
fenlei= soup.find('span', attrs={'style': '#2e2a2b'})  
print(laji.string)   
print(shuyu.string.strip())   
print(fenlei.string)  
else:  
print(**"**没有与此物品名称匹配的词条")

6.GUI

#此部分为GUI界面设计

class Classification(object):

def \_\_init\_\_(self):

#创建主窗口,用于容纳其他组件

self.root = tkinter.Tk()

#给主窗口设置标题内容

self.root.title("智能垃圾分类")

#创建一个输入框,并设置尺寸

self.trash\_input = tkinter.Entry(self.root,width=30,text="请输入垃圾的名称")

self.trash\_input.grid(row=0,column=1)

#创建一个回显列表

self.display\_info = tkinter.Listbox(self.root, width=50,height=10)

self.display\_info.grid(row=10, columnspan=8, sticky=tkinter.E)

#创建查询按钮

self.result\_button1 = tkinter.Button(self.root, command = self.waste\_sorting1, text = "文字查询")

self.result\_button1.grid(row=0,column=0)

self.result\_button2 = tkinter.Button(self.root, command = self.waste\_sorting2, text = "图片查询")

self.result\_button2.grid(row=1,column=0)

#完成布局

16.3.2 移动端微信小程序

1.baiduacesstoken.js

const rq = require('request-promise')

/\*获取百度ai AccessToken\*/

exports.main = async(event, context) => {

  let apiKey = 'wPnvuS8WCKeAj6OHfGGQlY3R',

    grantType = 'client\_credentials',

    secretKey = 'ljoNDsp2HIlb0ePwgQtGxwNxoW9idak4',

    url = `https://aip.baidubce.com/oauth/2.0/token`

  return new Promise(async(resolve, reject) => {

    try {

      let data = await rq({

        method: 'POST',

        url,

        form: {

          "grant\_type": grantType,

          "client\_secret": secretKey,

          "client\_id": apiKey

        },

        json: true

      })

      resolve({

        code: 0,

        data,

        info: '操作成功！'

      })

    } catch (error) {

      console.log(error)

      if (!error.code) reject(error)

      resolve(error)

    }

  })

}

2.baidu-token-util.js

本模块解决accesstoken每隔一个月就失效的问题，如果accesstoken不存在就重新申请。

const getBdAiAccessToken = function () {

    return  new Promise((resolve, reject) => {

        console.log('getBdAiAccessToken!');

        var time = wx.getStorageSync("time");

        var curTime = new Date().getTime();

        console.log('time:'+time+'----curTime:'+curTime);

        console.log(parseInt((curTime - time) / 1000/60/60/24));

        var timeNum = parseInt((curTime - time) / 1000/60/60/24);

        console.log("token生成天数timeNum:" + timeNum);

        var accessToken = wx.getStorageSync("access\_token")

        console.log("缓存中的accessToken===" + accessToken)

        if (timeNum > 28 || (accessToken == "" ||

            accessToken == null || accessToken == undefined)) {

            //token超过28天或者不存在，则调用云函数重新获取

            wx.cloud.callFunction({

                name: 'baiduAccessToken',

                success: res => {

                    console.log("云函数获取token:" + JSON.stringify(res))

                    var access\_token = res.result.data.access\_token

                    wx.setStorageSync("access\_token", access\_token);

                    wx.setStorageSync("time", new Date().getTime());

                    resolve(

                        {

                            'access\_token': access\_token

                        }

                    );

                },

                fail: error => {

                    console.error('[云函数] [sum] 调用失败：', error);

                    reject('调用云函数失败：' + JSON.stringify(error));

                }

            });

        } else {

            //缓存中存在有效的token

            resolve(

                {

                    'access\_token': accessToken

                }

            );

        }

    });

}

module.exports = {

    getBdAiAccessToken: getBdAiAccessToken,

}

3.camera.js

var http = require('../../../utils/http.js')

var baiduTokenUtil = require('../../../utils/baidu-token-util.js');

// import { Utilaa } from 'util'

// var u = require('underscore')

Page({

    data: {

        isShow: false,

        results: [],

        src: "",

        isCamera: true,

        btnTxt: "拍照"

    },

    accessToken: "",

    onLoad() {

        this.ctx = wx.createCameraContext();

        var that=this

        wx.showShareMenu({

            withShareTicket: true //要求小程序返回分享目标信息

        });

        try {

            baiduTokenUtil.getBdAiAccessToken().then(

                function (res) {

                    console.log('获取百度ai token:' + JSON.stringify(res));

                    that.accessToken = res.access\_token;

                }, function (error) {

                    console.error('获取百度ai token:' + error);

                }

            );

        } catch (error) {

            console.error(error);

        }

    },

    takePhoto() {

        var that = this

        if (this.data.isCamera == false) {

            this.setData({

                isCamera: true,

                btnTxt: "拍照"

            })

            return

        }

        this.ctx.takePhoto({

            quality: 'high',

            success: (res) => {

                this.setData({

                    src: res.tempImagePath,

                    isCamera: false,

                    btnTxt: "重拍"

                })

                wx.showLoading({

                    title: '正在加载中',

                })

                wx.getFileSystemManager().readFile({

                    filePath: res.tempImagePath,

                    encoding: "base64",

                    success: res => {

                        that.req(that.accessToken, res.data)

                    },

                    fail: res => {

                        wx.hideLoading()

                        wx.showToast({

                            title: '拍照失败,未获取相机权限或其他原因',

                            icon: "none"

                        })

                    }

                })

            }

        })

    },

    req: function (token, image) {

        var that = this

        var data = {

            "image": image

        }

        wx.request({

            url: 'https://aip.baidubce.com/rest/2.0/image-classify/v2/advanced\_general?access\_token=' + token,

            method: 'post',

            data: data,

            header: {

                "content-type": "application/x-www-form-urlencoded",

            },

            success (res) {

                wx.hideLoading();

                console.log(res.data)

                var results = res.data.result;

                if (results) {

                    that.setData({

                        isShow: true,

                        results: results

                    })

                } else {

                    wx.showToast({

                        icon: 'none',

                        title: '没有认出来，可以再试试~',

                    })

                }

            },

            fail(error){

                wx.hideLoading();

                console.log(error);

                wx.showToast({

                    icon: 'none',

                    title: '请求失败了，请确保网络正常，重新试试~',

                })

            }

        });

    },

    radioChange: function (e) {

        console.log(e)

        console.log(e.detail)

        console.log(e.detail.value)

        wx.navigateTo({

            url: '/pages/ai/search?searchText=' + e.detail.value,

        })

    },

    hideModal: function () {

        this.setData({

            isShow: false,

        })

    },

    error(e) {

        console.log(e.detail)

    }

})

4.index.js(录音)

此模块功能为识别录音并返回搜索结果。

var checkPermissionUtil = require('../../utils/check-permission-util.js');

var baiduTokenUtil = require('../../utils/baidu-token-util.js');

Page({

    data: {

        SHOW\_TOP: true,

        canRecordStart: false,

    },

    isSpeaking: false,

    accessToken: "",

    onLoad: function (options) {

        console.log("onLoad！");

        var that=this

        wx.showShareMenu({

            withShareTicket: true //要求小程序返回分享目标信息

        });

        var isShowed = wx.getStorageSync("tip");

        if (isShowed != 1) {

            setTimeout(() => {

                this.setData({

                    SHOW\_TOP: false

                })

                wx.setStorageSync("tip", 1)

            }, 3 \* 1000)

        } else {

            this.setData({

                SHOW\_TOP: false

            })

        };

        try {

            baiduTokenUtil.getBdAiAccessToken().then(

                function (res) {

                    console.log('获取百度ai token:' + JSON.stringify(res));

                    console.log(res.access\_token)

                    that.accessToken = res.access\_token ;

                }, function (error) {

                    console.error('获取百度ai token:' + error);

                }

            );

        } catch (error) {

            console.error(error);

        }

    },

    goSearch: function () {

        wx.navigateTo({

            url: '/pages/ai/search'

        });

    },

    onBindCamera: function () {

        console.log('onBindCamera!');

        var that = this;

        try {

   checkPermissionUtil.checkPermission('scope.camera').then(function (res) {

                    console.log('检测权限结果：' + res);

                    wx.navigateTo({

                        url: 'camera/camera',

                    });

                }, function (err) {

                    console.error('检测权限结果失败：' + err);

                    wx.showToast({

                        title: '授权失败，无法使用该功能~',

                        icon: 'none'

                    });

                }

            );

        } catch (err) {

            console.error(err);

            wx.showToast({

                title: '授权失败，无法使用该功能~',

                icon: 'none'

            });

            return

        }

    },

    onTouchStart: function () {

        console.log('onTouchStart!' + this.data.canRecordStart);

        speaking.call(this);

        this.setData({

            canRecordStart: true

        });

        this.startRecordHandle();

    },

    onTouchEnd: function () {

        console.log('onTouchEnd!canRecordStart:' + this.data.canRecordStart + '----isSpeaking:' + this.isSpeaking);

        clearInterval(this.timer);

        this.setData({

            canRecordStart: false

        });

        if (this.isSpeaking) {

            wx.getRecorderManager().stop();

        }

    },

    //录音前检测scope.record授权情况

    async startRecordHandle() {

        var that = this;

        try {

            await checkPermissionUtil.checkPermission('scope.record').then(function (res) {

                    console.log('检测权限结果：' + res);

                    that.record();

                }, function (err) {

                    console.error('检测权限结果失败：' + err);

                    wx.showToast({

                        title: '授权失败，无法使用该功能~',

                        icon: 'none'

                    });

                }

            );

        } catch (err) {

            console.error(err);

            wx.showToast({

                title: '授权失败，无法使用该功能~',

                icon: 'none'

            });

            return

        }

    },

    //开始录音的时候

    record: function () {

        var that = this;

        console.log('startRecord!');

        const recorderManager = wx.getRecorderManager();

        const options = {

            duration: 30000,//指定录音的时长，单位 ms

            sampleRate: 16000,//采样率

            numberOfChannels: 1,//录音通道数

            encodeBitRate: 48000,//编码码率

            format: 'aac',//音频格式，有效值aac/mp3

        };

     console.log('开始正式录音前，canRecordStart' + this.data.canRecordStart);

        //开始录音

        if (this.data.canRecordStart) {

            recorderManager.start(options);

            this.isSpeaking = true;

        }

        recorderManager.onStart(() => {

            console.log('recorder start')

        });

        recorderManager.onPause(() => {

            console.log('recorder pause')

        })

        recorderManager.onStop((res) => {

            this.isSpeaking = false;

            console.log('recorder stop', res);

            //wx.hideLoading();

            if (res && res.duration < 1000) {

                wx.showToast({

                    title: '说话时间太短啦！',

                    icon: 'none'

                })

                return;

            }

            if (res && res.duration > 8000) {

                wx.showToast({

                    title: '说的有点长，可以精简点呀~',

                    icon: 'none'

                })

                return;

            }

            const {tempFilePath} = res

            this.speechRecognition(res);

        })

        //错误回调

        recorderManager.onError((res) => {

            // wx.showToast({

            //     title: '录音出错啦，请重试！',

            //

            // });

            console.error('录音错误回调：' + JSON.stringify(res));

        })

    },

    speechRecognition: function (res) {

        wx.showLoading({

            title: '识别中...',

        })

        var that = this;

        var fileSize = res.fileSize;

        var tempFilePath = res.tempFilePath;

        var format = 'pcm';

        if (tempFilePath) {

            format=tempFilePath.substring(tempFilePath.lastIndexOf('.')+ 1);

        }

        const fileSystemManager = wx.getFileSystemManager()

        fileSystemManager.readFile({

            filePath: res.tempFilePath,

            encoding: "base64",

            success(res){

                console.log(res);

                var base64 = res.data;

                var data = {

                    "format": format,

                    "rate": 16000,

                    "dev\_pid": 80001,

                    "channel": 1,

                    "token": that.accessToken,

                    "cuid": "baidu\_workshop",

                    "len": fileSize,

                    "speech": base64

                }

                console.log('语音识别请求参数：' + JSON.stringify(data));

                wx.request({

                    url: 'https://vop.baidu.com/pro\_api',

                    method: 'post',

                    data: data,

                    success (res) {

                        wx.hideLoading();

                        console.log(res.data)

                        var result = res.data.result;

                        if (result && result.length > 0) {

                            var location = result[0].lastIndexOf("。");

                            var text = '';

                            console.log(result[0]);

                            console.log('符号位置：' + location);

                            text = result[0].replace(/[\ |\~|\`|\!|\@|\#|\$|\%|\^|\&|\\*|\(|\)|\-|\\_|\+|\=|\||\\|\[|\]|\{|\}|\;|\:|\"|\'|\,|\<|\.|\。|\，|\！|\；|\>|\/|\?]/g, "");

                            console.log('text' + text);

                            wx.navigateTo({

                                url: '/pages/ai/search?searchText=' + text

                            })

                        } else {

                            //没有result，认为语音识别失败

                            wx.showModal({

                                title: '提示',

                                content: '不知道你说的啥，可以再试试~',

                                showCancel: false,

                                success (res) {

                                    if (res.confirm) {

                                        console.log('用户点击确定')

                                    } else if (res.cancel) {

                                        console.log('用户点击取消')

                                    }

                                }

                            })

                        }

                    },

                    fail(error){

                        wx.hideLoading();

                        console.log(error);

                        wx.showToast({

                            icon: 'none',

                            title: '请求失败了，请确保网络正常，重新试试~',

                        })

                    }

                })

            },

            fail(res){

                wx.hideLoading();

                console.log(res)

            }

        })

    },

});

//麦克风帧动画

function speaking() {

    var \_this = this;

    //话筒帧动画

    var i = 1;

    this.timer = setInterval(function () {

        i++;

        i = i % 5;

        \_this.setData({

            j: i

        })

    }, 200);

}

5.garbage-sort-data.js（数据库）

建立数据库进行数据存储，数据库内容为各类垃圾常见类型。

var garbage\_sort\_data = [

    {

  "categroy": 1,

  "data": [{

    "letter":"A",

    "garbageItem": ["A4纸", "安全帽"]

  }, {

    "letter": "B",

      },

项目17 协同过滤音乐推荐系统

17.3 模块实现

17.3.1 数据预处理

1.数据集介绍

2.数据处理

相关代码如下：

#读取数据集

triplet\_dataset=pd.read\_csv(filepath\_or\_buffer=data\_home+'train\_triplets.txt', sep='\t', header=None, names=['user','song','play\_count'])

#对每一个用户，分别统计其播放量

output\_dict = {}

with open(data\_home+'train\_triplets.txt') as f:

for line\_number, line in enumerate(f):

#找到当前的用户

user = line.split('\t')[0]

#得到其播放量数据

play\_count = int(line.split('\t')[2])

#如果字典中已经有该用户信息，在其基础上增加当前的播放量

if user in output\_dict:

play\_count +=output\_dict[user]

output\_dict.update({user:play\_count})

output\_dict.update({user:play\_count})

#统计用户总播放量

output\_list = [{'user':k,'play\_count':v} for k,v in output\_dict.items()]

#转换成DF格式

play\_count\_df = pd.DataFrame(output\_list)

#排序

play\_count\_df = play\_count\_df.sort\_values(by = 'play\_count', ascending = False)

#输出为表格并保存

play\_count\_df.to\_csv(path\_or\_buf='user\_playcount\_df.csv', index = False)

#输出表格并保存，统计歌曲播放量

#对每首歌，统计其播放量，方法跟上述类似

output\_dict = {}

with open(data\_home+'train\_triplets.txt') as f:

for line\_number, line in enumerate(f):

#找到当前歌曲

song = line.split('\t')[1]

#找到当前播放次数

play\_count = int(line.split('\t')[2])

#统计每首歌曲被播放的总次数

if song in output\_dict:

play\_count +=output\_dict[song]

output\_dict.update({song:play\_count})

output\_dict.update({song:play\_count})

output\_list = [{'song':k,'play\_count':v} for k,v in output\_dict.items()]

#转换成DF格式

song\_count\_df = pd.DataFrame(output\_list)

song\_count\_df = song\_count\_df.sort\_values(by = 'play\_count', ascending = False)

#输出保存

#10万名用户的播放量占总体的比例

total\_play\_count = sum(song\_count\_df.play\_count)

print ((float(play\_count\_df.head(n=100000).play\_count.sum())/total\_play\_count)\*100)

play\_count\_subset = play\_count\_df.head(n=100000)

#3万首歌曲的播放量占总体的比例

(float(song\_count\_df.head(n=30000).play\_count.sum())/total\_play\_count)\*100

#取10万个用户，3万首歌

user\_subset = list(play\_count\_subset.user)

song\_subset = list(song\_count\_subset.song)

#只保留10万名用户的数据，其余过滤掉

triplet\_dataset\_sub=triplet\_dataset[triplet\_dataset.user.isin(user\_subset) ]

del(triplet\_dataset)

#只保留3万首歌曲的数据，其余过滤掉

triplet\_dataset\_sub\_song=triplet\_dataset\_sub[triplet\_dataset\_sub.song.isin(song\_subset)]

del(triplet\_dataset\_sub)

#加入音乐详细信息，合并两个数据集

conn = sqlite3.connect(data\_home+'track\_metadata.db')

cur = conn.cursor()

cur.execute("SELECT name FROM sqlite\_master WHERE type='table'")

cur.fetchall()

track\_metadata\_df = pd.read\_sql(con=conn, sql='select \* from songs')

track\_metadata\_df\_sub= track\_metadata\_df[track\_metadata\_df.song\_id.isin(song\_subset)]

track\_metadata\_df\_sub.to\_csv(path\_or\_buf=data\_home+'track\_metadata\_df\_sub.csv', index=False)

#去掉无用信息

del(track\_metadata\_df\_sub['track\_id'])

del(track\_metadata\_df\_sub['artist\_mbid'])

#去掉重复的

track\_metadata\_df\_sub = track\_metadata\_df\_sub.drop\_duplicates(['song\_id'])

#将音乐信息数据和之前的播放数据整合到一起

triplet\_dataset\_sub\_song\_merged = pd.merge(triplet\_dataset\_sub\_song, track\_metadata\_df\_sub, how='left', left\_on='song', right\_on='song\_id')

#可以自己改变列名

triplet\_dataset\_sub\_song\_merged.rename(columns={'play\_count':'listen\_count'},inplace=True)

#去掉不需要的指标

del(triplet\_dataset\_sub\_song\_merged['song\_id'])

del(triplet\_dataset\_sub\_song\_merged['artist\_id'])

del(triplet\_dataset\_sub\_song\_merged['duration'])

del(triplet\_dataset\_sub\_song\_merged['artist\_familiarity'])

del(triplet\_dataset\_sub\_song\_merged['artist\_hotttnesss'])

del(triplet\_dataset\_sub\_song\_merged['track\_7digitalid'])

del(triplet\_dataset\_sub\_song\_merged['shs\_perf'])

del(triplet\_dataset\_sub\_song\_merged['shs\_work'])

17.3.2 算法实现

1.基于流行度的推荐

相关代码如下：

#按歌曲名字统计其播放量的总数

popular\_songs=triplet\_dataset\_sub\_song\_merged[['title','listen\_count']].groupby('title').sum().reset\_index()

#对结果进行排序，展示播放数量位于前20的歌曲

popular\_songs\_top\_20=popular\_songs.sort\_values('listen\_count',ascending=False).head(n=20)

#转换成list格式方便画图

objects = (list(popular\_songs\_top\_20['title']))

#设置位置

y\_pos = np.arange(len(objects))

#对应结果值

performance = list(popular\_songs\_top\_20['listen\_count'])

#绘图

plt.bar(y\_pos, performance, align='center', alpha=0.5)

plt.xticks(y\_pos, objects, rotation='vertical')

plt.ylabel('播放量')

plt.title('最受欢迎歌曲')

plt.show()

2)生成排行榜单

相关代码如下：

def create\_popularity\_recommendation(train\_data, user\_id, item\_id):

#根据指定的特征统计播放情况，可以选择歌曲名、专辑名、歌手名

train\_data\_grouped = train\_data.groupby([item\_id]).agg({user\_id: 'count'}).reset\_index()

#用得分表示结果

train\_data\_grouped.rename(columns = {user\_id: 'score'},inplace=True)

#根据得分进行排序

train\_data\_sort = train\_data\_grouped.sort\_values(['score', item\_id], ascending = [0,1])

#加入一项排行等级，表示其推荐的优先级

train\_data\_sort['Rank'] = train\_data\_sort['score'].rank(ascending=0, method='first')

#返回指定个数的推荐结果

popularity\_recommendations = train\_data\_sort.head(20)

return popularity\_recommendations

#得到推荐结果

recommendations=create\_popularity\_recommendation(triplet\_dataset\_sub\_song\_merged,'user','title')

recommendations

2.基于物品的协同过滤推荐

相关代码如下：

1)对每首歌曲播放情况进行统计

#给定用户，找出用户听过的所有歌曲

def get\_user\_items(self, user):

user\_data = self.train\_data[self.train\_data[self.user\_id] == user]

user\_items = list(user\_data[self.item\_id].unique())

return user\_items

#给定歌曲，找出听过这首歌的所有用户

def get\_item\_users(self, item):

item\_data = self.train\_data[self.train\_data[self.item\_id] == item]

item\_users = set(item\_data[self.user\_id].unique())

return item\_users

#对数据集中的歌曲去重

def get\_all\_items\_train\_data(self):

all\_items = list(self.train\_data[self.item\_id].unique())

return all\_items

2)计算歌曲相似度，构建矩阵

相关代码如下：

#构建相似度矩阵

def construct\_cooccurence\_matrix(self, user\_songs, all\_songs):

user\_songs\_users = []

for i in range(0, len(user\_songs)):

user\_songs\_users.append(self.get\_item\_users(user\_songs[i]))

#设置矩阵大小为某一指定用户听过的所有歌曲×数据集中歌曲总数

cooccurence\_matrix = np.matrix(np.zeros(shape=(len(user\_songs), len(all\_songs))), float)

for i in range(0, len(all\_songs)):

#找出用户听过的第i首歌被哪些人听过

songs\_i\_data = self.train\_data[self.train\_data[self.item\_id] == all\_songs[i]]

users\_i = set(songs\_i\_data[self.user\_id].unique())

#找出歌曲集中第j首歌被哪些人听过

for j in range(0, len(user\_songs)):

users\_j = user\_songs\_users[j]

#计算听过i歌曲人数和j歌曲人数的交集

users\_intersection = users\_i.intersection(users\_j)

#采用Jaccard系数计算相似度

if len(users\_intersection) != 0:

#计算听过i歌曲人数和j歌曲人数的并集

users\_union = users\_i.union(users\_j)

#使用Jaccard系数计算i,j之间的相似度

cooccurence\_matrix[j,i] = float(len(users\_intersection)) / float(len(users\_union))

else:

cooccurence\_matrix[j,i] = 0

#改进之后相似度计算

if len(users\_intersection) != 0:

for k in users\_intersection:

user\_k\_data = self.get\_user\_items(k)

cooccurence\_matrix[j, i] += 1/math.log(1 + len(user\_k\_data)\*1.0)

cooccurence\_matrix[j, i] = float(cooccurence\_matrix[j, i]/math.sqrt(len(users\_i)\*len(users\_j)))

else:

cooccurence\_matrix[j, i] = 0

coo\_max = cooccurence\_matrix.max(axis=1)

cooccurence\_matrix = cooccurence\_matrix/coo\_max

return cooccurence\_matrix

3)根据相似度矩阵进行topN推荐

相关代码如下：

def generate\_top\_recommendations(self, user, cooccurence\_matrix, all\_songs, user\_songs):

print("Non zero values in cooccurence\_matrix :%d" % np.count\_nonzero(cooccurence\_matrix))

#对每一首待推荐歌曲，计算其与用户听过的所有歌曲相似度的平均值

user\_sim\_scores = cooccurence\_matrix.sum(axis=0) / float(cooccurence\_matrix.shape[0])

user\_sim\_scores = np.array(user\_sim\_scores)[0].tolist()

sort\_index = sorted(((e, i) for i, e in enumerate(list(user\_sim\_scores))), reverse=True)

columns = ['user\_id', 'song', 'score', 'rank']

df = pandas.DataFrame(columns=columns)

#推荐相似度最高的5首歌

rank = 1

for i in range(0, len(sort\_index)):

if ~np.isnan(sort\_index[i][0]) and all\_songs[sort\_index[i][1]] not in user\_songs and rank <= 5:

df.loc[len(df)] = [user, all\_songs[sort\_index[i][1]], sort\_index[i][0], rank]

rank = rank + 1

if df.shape[0] == 0:

print("The current user has no songs for training the item similarity based recommendation model.")

return -1

else:

return df

17.3.3 算法测评

具体实现

#k折交叉验证，这里选择将数据集分成5份，每份轮流作为验证集，其余作为训练集  
train\_data=pd.read\_csv(filepath\_or\_buffer='D:\\jupyter\\music\\triplet\_dataset\_sub\_song\_merged\_sub.csv',encoding = "ISO-8859-1")

is\_model = n.item\_similarity\_recommender\_py()

is\_model.create(train\_data, 'user', 'title')

all\_song = is\_model.get\_all\_items\_train\_data()

pre\_list = []

rec\_list = []

cov\_list = []

kf = KFold(n\_splits=5,shuffle=True, random\_state=5)

for i, (train\_index, test\_index) in enumerate(kf.split(train\_data)):

hit = 0  
 n\_pre = 0  
 n\_rec = 0  
 test = []

train = []

rec\_all = []

column = ['user\_id', 'song']

mydf = pd.DataFrame(columns=column)

df = pd.DataFrame(columns=column)

#使用k折划分出来的只是索引号，根据索引号找到所需数据，分别存入训练集、验证集  
 for j in test\_index:

u = list(train\_data.user)[j]

s = list(train\_data.song)[j]

#mydf = pd.DataFrame(columns=column)

#mm = pd.DataFrame([[u,s]])

#mydf = mydf.append(mm)

mydf.loc[len(mydf)] = [u, s]

if u not in test:

test.append(u)

for k in train\_index:

uu = list(train\_data.user)[k]

ss = list(train\_data.song)[k]

df.loc[len(df)] = [uu, ss]

if uu not in train:

train.append(uu)

#对训练集中每个用户进行推荐  
 for uid in train:

if uid not in test:

continue  
 else:

model\_train = n.item\_similarity\_recommender\_py()

model\_train.create(df, 'user\_id', 'song')

model\_test = n.item\_similarity\_recommender\_py()

model\_test.create(mydf,'user\_id', 'song')

#记录推荐结果  
 rec = model\_train.recommend(uid)

#在测试集中找出此推荐用户听过的歌

listen\_items = set(model\_test.get\_user\_items(uid))

rec\_item = set(rec['song'])

#找出推荐的和用户听过歌的交集

ht = len(listen\_items.intersection(rec\_item))

hit = ht + hit

#总推荐数

n\_pre = n\_pre + len(rec\_item)

#用户总计听过的歌

n\_rec = n\_rec + len(listen\_items)

#推荐了多少不同的歌

for item in list(rec.song):

if item not in rec\_all:

rec\_all.append(item)

#计算准确率、召回率、覆盖率

pre\_list.append(hit/(1.0\*n\_pre))

rec\_list.append(hit/(1.0\*n\_rec))

cov\_list.append(len(rec\_all)/len(all\_song))

#计算k次的结果取平均  
print(pre\_list,rec\_list,cov\_list)

print('准确率:', np.mean(pre\_list))

print('召回率:', np.mean(rec\_list))

('覆盖率:', np.mean(cov\_list))print('

项目18 护肤品推荐系统

18.3 模块实现

18.3.1 文件读入

相关代码如下：

#文件读入部分

user = pd.Series({'wxid':'o\_2phwQNVY9WYG1p0B1z0E\_d-lHM',

'T区油': 1,

'U区油': 1,

'敏感肌': 1,

'诉求': '祛痘',

'过敏成分': '烟酰胺'})

pro = pd.read\_csv(r'df\_product1046.csv', encoding='ANSI')

df\_component = pd.read\_csv("df\_component.csv",encoding='gb18030')

df\_fake = pd.read\_csv("df\_fake.csv",encoding="gb18030")

fformula = pd.read\_csv("Formula\_formatting.csv",encoding="gb18030")

ingredient\_banned = pd.read\_excel('ingredient\_banned\_to\_number.xlsx', encoding="gb18030")

18.3.2 推荐算法

1.数据预处理

相关代码如下：

def \_\_init\_\_(self, df\_fake, sub2\_product):

self.frame = df\_fake #调用文件

self.product = sub2\_product #产品表

#self.screened\_product\_path = r'D:\work\dataclinic\fake\df\_product1046.csv' #读取预筛选后的产品集

#self.\_init\_data()

#def \_init\_data(self):

#self.frame = pd.read\_csv(self.frame\_path)

#self.product = pd.read\_csv(self.product\_path,encoding='GB18030')

#self.screened\_product\_path = pd.read\_csv(self.product\_path,encoding='GB18030')

def screen(self, need): #数据预处理

self.frame = self.frame[(self.frame['诉求'].isin([need]))]

def vec\_purchase(self):

#提取购买记录并拉直

g = self.frame['购买记录']

g2 = self.frame['购买记录2']

g3 = self.frame['购买记录3']

wxid = list(self.frame['wechatid'])

s = pd.Series(wxid, index=g)

s2 = pd.Series(wxid, index=g2)

s3 = pd.Series(wxid, index=g3)

pin = pd.concat([s, s2, s3], axis=0) #数据合并

dict\_pin = {'wechatid': pin.values, '购买记录': pin.index, }

df2 = pd.DataFrame(dict\_pin)

#拉直后的dataframe（wechat id ：购买记录）

self.frame\_p = df2[~(df2['购买记录'].isin([-1]))]

2.计算相似度

相关代码如下：

#计算肤质向量(T区油、U区油、敏感肌、痘痘肌)的余弦相似度

def cosine\_skin(self, target\_user\_id, other\_user\_id):

#数据预处理

target\_skin = []

other\_skin = []

cols = ['T区油', 'U区油', '敏感肌', '痘痘肌']

for col in cols:

target\_skin.append((self.frame[self.frame['wechatid'] == target\_user\_id][col].values[0]) \* 2 - 1) #标准化可能

for col in cols:

other\_skin.append((self.frame[self.frame['wechatid'] == other\_user\_id][col].values[0]) \* 2 - 1)

#计算余弦相似度

nume=sum(np.multiply(np.array(target\_skin),np.array(other\_skin)))#分子

deno=sum(np.array(target\_skin)\*\* 2)\*sum(np.array(other\_skin)\*\* 2)#分母

cosine = nume / math.sqrt(deno) #值为1

return cosine

#计算购买记录余弦相似度

def cosine\_purchase(self, target\_user\_id, other\_user\_id):

target\_items = self.frame\_p[self.frame\_p['wechatid'] == target\_user\_id]['购买记录']

items = self.frame\_p[self.frame\_p['wechatid'] == other\_user\_id]['购买记录']

union\_len = len(set(target\_items) & set(items))

if union\_len == 0:

return 0.0

product = len(target\_items) \* len(items)

cosine = union\_len / math.sqrt(product)

return cosine

#计算加权平均相似度并排序

def get\_top\_n\_users(self, target\_user\_id, top\_n):

#提取其他所有用户

other\_users\_id = [i for i in set(self.frame\_p['wechatid']) if i != target\_user\_id]

#计算与其他用户的购买相似度

sim\_purchase\_list = [self.cosine\_purchase(target\_user\_id, other\_user\_id) for other\_user\_id in other\_users\_id]

#计算与其他用户的肤质相似度

sim\_skin\_list = [self.cosine\_skin(target\_user\_id, other\_user\_id) for other\_user\_id in other\_users\_id]

#加权平均（各占50%）

sim\_list = list((np.array(sim\_purchase\_list) + np.array(sim\_skin\_list)) / 2)

sim\_list = sorted(zip(other\_users\_id, sim\_list), key=lambda x: x[1], reverse=True)

return sim\_list[:top\_n]

3.排序并提取产品

相关代码如下：

#提取候选产品表

def get\_candidates\_items(self, target\_user\_id):

target\_user\_item = set(self.frame\_p[self.frame\_p['wechatid'] == target\_user\_id]['购买记录'])

other\_user\_item = set(self.frame\_p[self.frame\_p['wechatid'] != target\_user\_id]['购买记录'])

candidates\_item = other\_user\_item - target\_user\_item

#寻找候选推荐品标准：目标用户没有使用过的（必要性存疑）

candidates\_item = list(candidates\_item & set(self.product['ind'].values))

#候选推荐品必须属于上一步筛选出的项目(目前使用全产品表代替筛选后产品表)

return candidates\_item

#计算用户兴趣程度

def get\_top\_n\_items(self, top\_n\_users, candidates\_items, top\_n):

top\_n\_user\_data = [self.frame\_p[self.frame\_p['wechatid'] == k] for k, \_ in top\_n\_users]

interest\_list = []

for ind in candidates\_items:

tmp = []

for user\_data in top\_n\_user\_data:

if ind in user\_data['购买记录'].values:

tmp.append(1)

else:

tmp.append(0)

interest = sum([top\_n\_users[i][1] \* tmp[i] for i in range(len(top\_n\_users))])

interest\_list.append((ind, interest))

interest\_list = sorted(interest\_list, key=lambda x: x[1], reverse=True)

return interest\_list[:top\_n]

#输入wxid，需求默认推荐产品数为10 输出有序推荐产品

def calculate(self, target\_user):

top\_n = self.product.shape[0]

target\_user\_id = target\_user.wxid

need = target\_user.诉求

self.screen(need)

self.vec\_purchase()

top\_n\_users=self.get\_top\_n\_users(target\_user\_id, top\_n)

candidates\_items = self.get\_candidates\_items(target\_user\_id)

top\_n\_items = self.get\_top\_n\_items(top\_n\_users, candidates\_items, top\_n)

#重构数据格式返回完整推荐产品信息

productlist = [top\_n\_items[i][0] for i in range(len(top\_n\_items))]

product\_rec = self.product[(self.product['ind'].isin(productlist))]

product\_rec['InterestRate'] = [top\_n\_items[i][1] for i in range(len(top\_n\_items))]

return product\_rec

4.组合推荐算法

相关代码如下：

#组合推荐算法

class CombRating():

def \_\_init\_\_(self,user, pro\_withrate, fformula):

self.user = user

self.product = pro\_withrate

self.fformula=fformula

#第一个for 找到用户的诉求是哪一种，要求四个属性全部对上

#第二个for 找到组合中应当有的产品类型，水、乳、霜、祛痘凝胶、洁面

def find\_kind(self):

#print(self.fformula)

n\_formula = self.fformula.shape[0]

for i in range(n\_formula):

if (self.user.诉求 == self.fformula.诉求[i]) \

and (self.user.T区油 == self.fformula.T区油[i]) \

and (self.user.U区油 == self.fformula.U区油[i]) \

and (self.user.敏感肌 == self.fformula.敏感肌[i]):

i\_formula = i

break

#此处使用总共的产品种类解决数字问题

#寻找第一个是产品类型的列并记录此前经过的列数

form\_list = []

total\_pro\_type = ['水', '乳', '霜', '祛痘凝胶', '洁面']

type\_number = 0

for j in range(len(self.fformula.columns)):

if self.fformula.columns[j] in total\_pro\_type:

break

else:

type\_number = type\_number + 1

#再找到所有需要的产品种类

for j in range(type\_number, len(self.fformula.columns)):

if (self.fformula.loc[i\_formula][j] == 1):

form\_list.append(self.fformula.columns[j])

return form\_list

def outer\_multiple(self, form\_list):

ddict={}

for i in range(len(form\_list)):

ddict[form\_list[i]] = list(self.product[self.product.剂型 == form\_list[i]].ind)

#print(ddict)

dd = []

for i in itertools.product(\*ddict.values()):

dd.append(i)

comb\_pd = pd.DataFrame(dd)

#为DF的每一列添加名称

column\_name = []

for i in range(len(comb\_pd.columns)):

column\_name.append('产品'+str(i+1))

comb\_pd.columns = column\_name

#返回的是产品编号ind一列的值

return comb\_pd

18.3.3 应用模块

1.得到最终产品

相关代码如下：

#整合

class Recommendation():

def \_\_init\_\_(self, user, pro, df\_component, df\_fake, fformula, ingredient\_banned):

self.user = user

self.pro = pro

self.df\_component = df\_component

self.df\_fake = df\_fake

self.fformula = fformula

self.ingredient\_banned = ingredient\_banned

#诉求筛选得到sub1

def sub1\_product(self):

#通过用户筛选需求成分，返回筛选后的产品列表sub1

pro = self.pro

user = self.user

#T区条件筛选

if user['T区油'] == 1:

for index in pro.index:

if pro.loc[index, 'typeT区：油'] != 1:

pro = pro.drop(index=index)

elif user['T区油'] == 0:

for index in pro.index:

if pro.loc[index, 'typeT区：干'] != 1:

pro = pro.drop(index=index)

#U区条件筛选

if user['U区油'] == 1:

for index in pro.index:

if pro.loc[index, 'typeU区：油'] != 1:

pro = pro.drop(index=index)

elif user['U区油'] == 0:

for index in pro.index:

if pro.loc[index, 'typeU区：干'] != 1:

pro = pro.drop(index=index)

#敏感肌筛选

if user['敏感肌'] == 1:

for index in pro.index:

if pro.loc[index, '敏感'] != 1:

pro = pro.drop(index=index)

#诉求筛选美白/祛痘

if user['诉求'] == '祛痘':

for index in pro.index:

if pro.loc[index, '诉求'] != '祛痘':

pro = pro.drop(index=index)

elif user['诉求'] == '美白':

for index in pro.index:

if pro.loc[index, '诉求'] != '美白':

pro = pro.drop(index=index)

pro = pro.reset\_index(drop=True)

sub1 = pro

return sub1

2.筛选过敏物质

相关代码如下：

#过敏物质筛选，得到sub2

def sub2\_product(self):

#通过用户过敏成分筛选产品，得到sub2

user = self.user

product = self.sub1\_product()

#1从user信息中提取过敏成分

allergic\_cpnt = user['过敏成分']

#2选出含有过敏成分的产品

product\_allergic = []

for i in range(0, len(df\_component.成分)):

if df\_component.成分[i] == allergic\_cpnt:

product\_allergic.append(df\_component.ind[i])

#3-1 生成sub2产品表，筛除含有过敏成分的产品，返回sub2产品表

sub2\_product = pd.DataFrame()

sub2\_product = product[:]

for i in range(0, len(product.ind)):

if i in product\_allergic:

sub2\_product.drop(index=[i], inplace=True)

sub2 = sub2\_product

return sub2

#输入两个产品的ind 返回过敏信息用于后面函数的调用

def is\_pro\_component\_banned(self, pro1\_ind, pro2\_ind):

#输入两个产品的ind 产品成分表、成分禁忌表、总产品表

#根据产品ind判断是否过敏，并且返回禁忌成分的字符串

df\_component = self.df\_component

ingredient\_banned = self.ingredient\_banned

pro = self.pro

3.筛选相互禁忌的产品

相关代码如下：

#对禁忌表进行预处理

ingredient\_name = ingredient\_banned.columns

ingredient\_banned= ingredient\_banned.drop(ingredient\_banned.columns[0], axis=1) #删除第一列

ingredient\_banned.index = ingredient\_name #重置横标签为产品名

#找出两个产品中所有的成分存入两个列表

pro1\_component = []

pro2\_component = []

for index in range(len(df\_component.index)):

if df\_component.loc[index, 'ind'] == pro1\_ind:

pro1\_component.append(df\_component.loc[index, '成分'])

elif df\_component.loc[index, 'ind'] == pro2\_ind:

pro2\_component.append(df\_component.loc[index, '成分'])

#print(pro1\_component, pro2\_component)

#寻找是否冲突，并且记录成分、产品这一版先用字符串作为返回值

banned\_record = ''

for com1 in pro1\_component:

for com2 in pro2\_component:

if (com1 in ingredient\_banned.index) and (com2 in ingredient\_banned.index):

if ingredient\_banned.loc[com1, com2] == 2:

li1 = list(pro[pro.ind == pro1\_ind].typenickname)

li1 = ''.join(li1)

li2 = list(pro[pro.ind == pro2\_ind].typenickname)

li2 = ''.join(li2)

banned\_record = banned\_record + '产品' + li1 + '与产品' + li2 + '相互禁忌' + '禁忌成分为' + com1 + '与' + com2

elif ingredient\_banned.loc[com1, com2] == 1:

li1 = list(pro[pro.ind == pro1\_ind].typenickname)

li1 = ''.join(li1)

li2 = list(pro[pro.ind == pro2\_ind].typenickname)

li2 = ''.join(li2)

banned\_record = banned\_record + '产品' + li1 + '与产品' + li2 + '相互禁忌' + '禁忌成分为' + com1 + '与' + com2

return banned\_record

#输入推荐组合 调用前方函数返回最后有备注的组合推荐

def is\_comb\_banned(self, comb\_pd):

#传入信息为 is\_pro\_component\_banned 的参数加上推荐组合的df

#增加df一列，用以存贮禁忌信息，数据形式为str

#对每个组合进行循环，创建banned\_info列表

#对每两个产品调用 is\_pro\_component\_banned

#若存在禁忌信息加入上述str，将banned\_info加入df的新列

df\_component = self.df\_component

ingredient\_banned = self.ingredient\_banned

self.pro = self.pro

comb\_pd['禁忌搭配情况'] = None

#对每个组合

for index in range(len(comb\_pd.index)):

total\_banned = ''

#对每两个产品

for pro1 in range(len(comb\_pd.columns)):

for pro2 in range(pro1, len(comb\_pd.columns)):

banned = self.is\_pro\_component\_banned(comb\_pd.ix[index, pro1], comb\_pd.ix[index, pro2])

if banned != '':

total\_banned = total\_banned + banned

#将得到此列的禁忌信息加入整个pd并返回

comb\_pd.loc[index, '禁忌搭配情况'] = total\_banned

#comb\_pd.to\_csv('result')

return comb\_pd

4.输出单品推荐与组合推荐

相关代码如下：

#单品推荐

def single\_rec(self):

user = self.user

#调用User类进行推荐

sub2 = self.sub2\_product()

U1 = UserCF(self.df\_fake, sub2)

items = U1.calculate(self.user)

return items

#复合推荐缺少护肤公式

def combine\_rec(self):

user = self.user

#调用User类先进行单品推荐

sub2 = self.sub2\_product()

U1 = UserCF(self.df\_fake, sub2)

items = U1.calculate(self.user)

#再调用Comb类进行复合推荐

C1 = CombRating(user, items, self.fformula)

ddd = C1.outer\_multiple(C1.find\_kind())

#再调用禁忌类对此进行处理

return self.is\_comb\_banned(ddd)

18.3.4 测试调用函数

相关代码如下：

#测试代码1

R1 = Recommendation(user, pro, df\_component, df\_fake, fformula, ingredient\_banned)

#print(R1.combine\_rec(), R1.single\_rec())

a = R1.combine\_rec()

b = R1.single\_rec()

a.to\_csv("file1\_1")

b.to\_csv("file2\_1")

项目19 基于人脸识别的特定整蛊推荐系统

19.3 模块实现

19.3.1 人脸识别

1.模型图片的获取及数据分析

相关代码如下：

#功能：用户上传图片

root = tkinter.Tk() #创建Tkinter.Tk()实例  
root.withdraw() #将Tkinter.Tk()实例隐藏  
path = os.getcwd()+"/face-recognition" #获取该程序文件目录的路径  
if not os.path.exists(path):

#判断文件夹是否存在，不存在则新建文件夹，存在进行下一步  
 os.mkdir(path)  
#打开Windows选择上传图片  
f\_name = tkinter.filedialog.askopenfilename(title=u'上传', filetypes=[("JPEG", ".jpg"), ("PNG", ".png")])  
shutil.copy(f\_name, path)

对用户上传的图片进行数据分析，提取人脸特征，作为下一步人脸识别的比对数据，相关代码如下：

known\_face\_encoding = [] #模型图片人脸数据列表

#功能：读取分析模型图片的人脸数据  
for fn in os.listdir(path):  
 #测试时，打印读取过的图片名字，确保遍历图片，实际应用时可注释掉  
 print(path + "/" + fn)  
 #得到人脸数据并放入列表  
 known\_face\_encoding.append(  
 face\_recognition.face\_encodings(  
 face\_recognition.load\_image\_file(path + "/" + fn))[0])

2.人脸检测及识别

相关代码如下：

pic\_show = tkinter.Tk() #创建一个Tkinter.Tk()实例  
pic\_show.withdraw() #将Tkinter.Tk()实例隐藏  
pic\_name = tkinter.filedialog.askopenfilename(title=u'打开', filetypes=[("JPG", ".jpg"), ("PNG", ".png")])  
picture = cv2.imread(pic\_name)  
#发现在图片中所有的脸和面部特征  
f\_locations = face\_recognition.face\_locations(picture)  
f\_encodings = face\_recognition.face\_encodings(picture, f\_locations)  
#在图片中循环遍历每个人脸  
for (top, right, bottom, left), face\_encoding in zip(f\_locations, f\_encodings):  
#对视频中一个人脸的比对结果（可能比对人脸库中多个人脸）  
 match = face\_recognition.compare\_faces(known\_face\_encoding, face\_encoding, tolerance=0.5)  
 if True in match:  
#测试时，画出一个绿框，框住脸（在进行美颜时该语句可注释掉）  
 cv2.rectangle(picture,(left, top),(right, bottom),(0, 255, 0), 2)  
#在此添加进行人像美颜的代码  
 else:  
#测试时，画出一个红框，框住脸（在进行美颜时该语句可注释掉）  
 cv2.rectangle(picture, (left, top), (right, bottom), (0, 0, 255), 2)  
#进行整蛊“美颜”

从摄像头获取图像，检测并识别人脸，相关代码如下：

cap = cv2.VideoCapture(700) #配置摄像头  
while True:  
 ret, frame = cap.read() #打开摄像头并获取画面帧  
#发现在该视频帧中所有的脸和face\_encodings  
 face\_locations = face\_recognition.face\_locations(frame)  
 face\_encodings = face\_recognition.face\_encodings(frame, face\_locations)  
#在这个视频帧中循环遍历每个人脸  
 for (top, right, bottom, left), face\_encoding in zip(face\_locations, face\_encodings):  
#对视频中一个人脸的比对结果（可能比对人脸库中多个人脸）  
 match = face\_recognition.compare\_faces(known\_face\_encoding, face\_encoding, tolerance=0.5)  
 if True in match:  
 #测试时，画出一个绿框，框住脸（在进行美颜时该语句可注释掉）  
 cv2.rectangle(frame, (left, top), (right, bottom), (0, 255, 0), 2)  
 #在此添加进行人像美颜的代码  
 else:  
 #测试时，画出一个红框，框住脸（在进行美颜时该语句可注释掉）  
 cv2.rectangle(frame,(left, top),(right, bottom),(0, 0, 255), 2)  
 #进行整蛊“美颜”

19.3.2 美颜处理

1.获取人脸五官切片

class NoFace(Exception):

#没有人脸

pass

class Organ():

def \_\_init\_\_(self, im\_bgr, im\_hsv, temp\_bgr, temp\_hsv, landmark, name, ksize=None):

#五官部位类，参数如下：

#im\_bgr：uint8数组，BGR图像的推断

#im\_hsv：uint8数组，HSV图像的推断

#temp\_bgr/hsv：全局临时映像

#landmark：array（x，2），地标

#name：字符串

#ksize：尺寸

self.im\_bgr, self.im\_hsv, self.landmark, self.name = im\_bgr, im\_hsv, landmark, name

self.get\_rect()

self.shape = (int(self.bottom - self.top), int(self.right - self.left))

self.size = self.shape[0] \* self.shape[1] \* 3

self.move = int(np.sqrt(self.size / 3) / 20)

self.ksize = self.get\_ksize()

self.patch\_bgr, self.patch\_hsv = self.get\_patch(self.im\_bgr), self.get\_patch(self.im\_hsv)

self.set\_temp(temp\_bgr, temp\_hsv)

self.patch\_mask = self.get\_mask\_re()

pass

def set\_temp(self, temp\_bgr, temp\_hsv):

self.im\_bgr\_temp, self.im\_hsv\_temp = temp\_bgr, temp\_hsv

self.patch\_bgr\_temp, self.patch\_hsv\_temp= self.get\_patch(self.im\_bgr\_temp), self.get\_patch(self.im\_hsv\_temp)

def confirm(self):

#确认操作

self.im\_bgr[:], self.im\_hsv[:] = self.im\_bgr\_temp[:], self.im\_hsv\_temp[:]

def update\_temp(self):

#更新临时图片

self.im\_bgr\_temp[:], self.im\_hsv\_temp[:] = self.im\_bgr[:], self.im\_hsv[:]

def get\_ksize(self, rate=15):

size = max([int(np.sqrt(self.size / 3) / rate), 1])

size = (size if size % 2 == 1 else size + 1)

return (size, size)

def get\_rect(self):

#获得定位方框

ys, xs = self.landmark[:, 1], self.landmark[:, 0]

self.top, self.bottom, self.left, self.right = np.min(ys), np.max(ys), np.min(xs), np.max(xs)

def get\_patch(self, im):

#截取局部切片

shape = im.shape

return im[np.max([self.top - self.move, 0]):np.min([self.bottom + self.move, shape[0]]),

np.max([self.left - self.move, 0]):np.min([self.right + self.move, shape[1]])]

def \_draw\_convex\_hull(self, im, points, color):

#勾画多凸边形

points = cv2.convexHull(points)

cv2.fillConvexPoly(im, points, color=color)

def get\_mask\_re(self, ksize=None):

#获得局部相对坐标遮罩

if ksize == None:

ksize = self.ksize

landmark\_re = self.landmark.copy()

landmark\_re[:, 1] -= np.max([self.top - self.move, 0])

landmark\_re[:, 0] -= np.max([self.left - self.move, 0])

mask = np.zeros(self.patch\_bgr.shape[:2], dtype=np.float64)

self.\_draw\_convex\_hull(mask,

landmark\_re,

color=1)

mask = np.array([mask, mask, mask]).transpose((1, 2, 0))

mask = (cv2.GaussianBlur(mask, ksize, 0) > 0) \* 1.0

return cv2.GaussianBlur(mask, ksize, 0)[:]

def get\_mask\_abs(self, ksize=None):

#获得全局绝对坐标遮罩

if ksize == None:

ksize = self.ksize

mask = np.zeros(self.im\_bgr.shape, dtype=np.float64)

patch = self.get\_patch(mask)

patch[:] = self.patch\_mask[:]

return mask

2.分类美颜操作

def whitening(self, rate=0.15, confirm=True):

#提亮美白

arguments:

rate:float,-1~1,new\_V=min(255,V\*(1+rate))

confirm:wether confirm this option

if confirm:

self.confirm()

self.patch\_hsv[:, :, -1] = np.minimum(

self.patch\_hsv[:, :, -1] + self.patch\_hsv[:, :, -1] \* self.patch\_mask[:, :, -1] \* rate, 255).astype(

'uint8')

self.im\_bgr[:] = cv2.cvtColor(self.im\_hsv, cv2.COLOR\_HSV2BGR)[:]

self.update\_temp()

else:

self.patch\_hsv\_temp[:] = cv2.cvtColor(self.patch\_bgr\_temp, cv2.COLOR\_BGR2HSV)[:]

self.patch\_hsv\_temp[:, :, -1] = np.minimum(

self.patch\_hsv\_temp[:, :, -1] + self.patch\_hsv\_temp[:, :, -1] \* self.patch\_mask[:, :, -1] \* rate,

255).astype('uint8')

self.patch\_bgr\_temp[:] = cv2.cvtColor(self.patch\_hsv\_temp, cv2.COLOR\_HSV2BGR)[:]

def brightening(self, rate=0.3, confirm=True):

#提升鲜艳度，参数

#rate:浮点型,-1~1,new\_S=min(255,S\*(1+rate))

#confirm:确认是否进行此选项

patch\_mask = self.get\_mask\_re((1, 1))

if confirm:

self.confirm()

patch\_new=self.patch\_hsv[:, :, 1]\* patch\_mask[:, :, 1] \* rate

patch\_new = cv2.GaussianBlur(patch\_new, (3, 3), 0)

self.patch\_hsv[:, :, 1] = np.minimum(self.patch\_hsv[:, :, 1] + patch\_new, 255).astype('uint8')

self.im\_bgr[:]=cv2.cvtColor(self.im\_hsv,cv2.COLOR\_HSV2BGR)[:]

self.update\_temp()

else:

self.patch\_hsv\_temp[:] = cv2.cvtColor(self.patch\_bgr\_temp, cv2.COLOR\_BGR2HSV)[:]

patch\_new=self.patch\_hsv\_temp[:,:,1]\*patch\_mask[:,:,1]\* rate

patch\_new = cv2.GaussianBlur(patch\_new, (3, 3), 0)

self.patch\_hsv\_temp[:, :, 1] = np.minimum(self.patch\_hsv[:, :, 1] + patch\_new, 255).astype('uint8')

self.patch\_bgr\_temp[:] = cv2.cvtColor(self.patch\_hsv\_temp, cv2.COLOR\_HSV2BGR)[:]

def smooth(self, rate=0.6, ksize=None, confirm=True):

#磨皮，参数

#rate:浮点型,0~1,im=rate\*new+(1-rate)\*src

#confirm:确认是否执行选项

if ksize == None:

ksize = self.get\_ksize(80)

index = self.patch\_mask > 0

if confirm:

self.confirm()

patch\_new = cv2.GaussianBlur(cv2.bilateralFilter(self.patch\_bgr, 3, \*ksize), ksize, 0)

self.patch\_bgr[index] = np.minimum(rate \* patch\_new[index] + (1 - rate) \* self.patch\_bgr[index],255).astype('uint8')

self.im\_hsv[:] = cv2.cvtColor(self.im\_bgr, cv2.COLOR\_BGR2HSV)[:]

self.update\_temp()

else:

patch\_new = cv2.GaussianBlur(cv2.bilateralFilter(self.patch\_bgr\_temp,3,\*ksize),ksize, 0)

self.patch\_bgr\_temp[index] = np.minimum(rate \* patch\_new[index] + (1 - rate) \* self.patch\_bgr\_temp[index],255).astype('uint8')

self.patch\_hsv\_temp[:] = cv2.cvtColor(self.patch\_bgr\_temp, cv2.COLOR\_BGR2HSV)[:]

def sharpen(self, rate=0.3, confirm=True):

#锐化

patch\_mask = self.get\_mask\_re((3, 3))

kernel = np.zeros((9, 9), np.float32)

kernel[4, 4] = 2.0 # Identity, times two!

#创建盒子滤波

boxFilter = np.ones((9, 9), np.float32) / 81.0

kernel = kernel - boxFilter

index = patch\_mask > 0

if confirm:

self.confirm()

sharp = cv2.filter2D(self.patch\_bgr, -1, kernel)

self.patch\_bgr[index] = np.minimum(((1 - rate) \* self.patch\_bgr)[index] + sharp[index] \* rate, 255).astype(

'uint8')

self.update\_temp()

else:

sharp = cv2.filter2D(self.patch\_bgr\_temp, -1, kernel)

self.patch\_bgr\_temp[:] = np.minimum(self.patch\_bgr\_temp + self.patch\_mask \* sharp \* rate, 255).astype(

'uint8')

self.patch\_hsv\_temp[:] = cv2.cvtColor(self.patch\_bgr\_temp, cv2.COLOR\_BGR2HSV)[:]

项目20 TensorFlow 2实现AI推荐换脸

20.3 模块实现

20.3.1 数据集

**1.数据载入**

!wget -nc "https://labfile.oss.aliyuncs.com/courses/1460/data.zip" #下载数据集

!unzip -o "data.zip" #解压

import os #遍历directory下的所有文件，并把路径用一个列表进行返回

def get\_image\_paths(directory):

return [x.path for x in os.scandir(directory) if x.name.endswith(".jpg") or x.name.endswith(".png")]

images\_A = get\_image\_paths("trump")

images\_B = get\_image\_paths("cage")

print("川普图片个数为 {}\n凯奇的图片个数为 {}".format(len(images\_A), len(images\_B)))

(2)使用Python中的OpenCV库，对图片进行批量加载。

import cv2

import numpy as np #批量加载图片，传入的是路径集合，遍历所有的路径，并加载图片

def load\_images(image\_paths):

iter\_all\_images = (cv2.imread(fn) for fn in image\_paths)

#iter\_all\_images 是一个generator类型，将它转换成熟知的Numpy的列表类型并返回

for i, image in enumerate(iter\_all\_images):

if i == 0: #对all\_images 进行初始,并且指定格式

all\_images = np.empty(

(len(image\_paths),) + image.shape, dtype=image.dtype)

all\_images[i] = image

return all\_images

**2.数据增强**

def random\_transform(image):

h, w = image.shape[0:2] #随机初始化旋转角度，范围-10~10之间

rotation = np.random.uniform(-10, 10) #随机初始化缩放比例，范围0.95~1.05

scale = np.random.uniform(0.95, 1.05) #随机定义平移距离，范围为-0.05~0.05

tx = np.random.uniform(-0.05, 0.05) \* w

ty = np.random.uniform(-0.05, 0.05) \* h #定义放射变化矩阵，整合之前参数变化

mat = cv2.getRotationMatrix2D((w//2, h//2), rotation, scale)

mat[:, 2] += (tx, ty)

#进行放射变化，根据变化矩阵中参数，将图片逐步变化，并返回变化后的图片

result = cv2.warpAffine(

image, mat, (w, h), borderMode=cv2.BORDER\_REPLICATE)

#图片有40%的可能性被翻转

if np.random.random() < 0.4:

result = result[:, ::-1]

return result

**3.构造Batch数据集**

def get\_training\_data(images, batch\_size):

#分批的同时把数据集打乱，有序的数据集可能使模型学偏

indices = np.random.randint(len(images), size=batch\_size)

for i, index in enumerate(indices):

#处理该批数据集

image = images[index]

#将图片进行预处理

image = random\_transform(image)

warped\_img, target\_img = random\_warp(image)

#开始分批

if i == 0:

warped\_images = np.empty(

(batch\_size,) + warped\_img.shape, warped\_img.dtype)

target\_images = np.empty(

(batch\_size,) + target\_img.shape, warped\_img.dtype)

warped\_images[i] = warped\_img

target\_images[i] = target\_img

return warped\_images, target\_images

20.3.2 自编码器

1.子像素卷积

#子像素卷积层，用于上采样

from keras.utils import conv\_utils

from keras.engine.topology import Layer

import keras.backend as K

class PixelShuffler(Layer): #初始化、子像素卷积层，并在输入数据时进行标准化处理

def \_\_init\_\_(self, size=(2, 2), data\_format=None, \*\*kwargs):

super(PixelShuffler, self).\_\_init\_\_(\*\*kwargs)

self.data\_format = K.normalize\_data\_format(data\_format)

self.size = conv\_utils.normalize\_tuple(size, 2, 'size')

def call(self, inputs): #根据得到输入层图层 batch\_size，h ，w，c 的大小

input\_shape = K.int\_shape(inputs)

batch\_size, h, w, c = input\_shape

if batch\_size is None:

batch\_size = -1

rh, rw = self.size #计算转换后的图层大小与通道数

oh, ow = h \* rh, w \* rw

oc = c // (rh \* rw)

#先将图层分开，并将每一层装换到自己应该到的维度

#最后再利用一次reshape()函数（计算机从外到里将数据逐个排序），转成指定大小的图层

out = K.reshape(inputs, (batch\_size, h, w, rh, rw, oc))

out = K.permute\_dimensions(out, (0, 1, 3, 2, 4, 5))

out = K.reshape(out, (batch\_size, oh, ow, oc))

return out

#compute\_output\_shape()函数用来输出这一层输出尺寸的大小

#尺寸是根据input\_shape以及定义output\_shape计算

def compute\_output\_shape(self, input\_shape):

height = input\_shape[1] \* self.size[0] if input\_shape[1] is not None else None

width = input\_shape[2] \* self.size[1] if input\_shape[2] is not None else None

channels = input\_shape[3] // self.size[0] // self.size[1]

return (input\_shape[0],

height,

width,

channels)

#设置配置文件

def get\_config(self):

config = {'size': self.size,

'data\_format': self.data\_format}

base\_config = super(PixelShuffler, self).get\_config()

return dict(list(base\_config.items()) + list(config.items()))

2.下采样层与上采样层

from keras.layers.advanced\_activations import LeakyReLU

from keras.layers.convolutional import Conv2D

#下采样层,filters为输出图层的通道数

#n\*n\*c->0.5n\*0.5n\*filters

def conv(filters):

def block(x) #每层由一个使图层大小减小一半的卷积层和一个LeakyReLU激活函数层构成

x = Conv2D(filters, kernel\_size=5, strides=2, padding='same')(x)

x = LeakyReLU(0.1)(x)

return x

return block

#上采样层，扩大图层大小

#图层的形状变化如下

#n\*n\*c->n\*n\*4filters->2n\*2n\*filters

def upscale(filters):

#每一层由一个扩大通道层的卷积，一个激活函数和一个像素洗牌层

def block(x):

#将通道数扩大为原来的四倍。为能够通过像素洗牌，使原来的图层扩大两倍

x = Conv2D(filters\*4, kernel\_size=3, padding='same')(x)

x = LeakyReLU(0.1)(x)

x = PixelShuffler()(x)

return x

return block

from keras.models import Model

from keras.layers import Input, Dense, Flatten, Reshape

#定义原图片的大小

IMAGE\_SHAPE = (64, 64, 3)

#定义全连接的神经元个数

ENCODER\_DIM = 1024

def Encoder():

input\_ = Input(shape=IMAGE\_SHAPE)

x = input\_

x = conv(128)(x)

x = conv(256)(x)

x = conv(512)(x)

x = conv(1024)(x)

x = Dense(ENCODER\_DIM)(Flatten()(x))

x = Dense(4\*4\*1024)(x)

x = Reshape((4, 4, 1024))(x)

x = upscale(512)(x)

return Model(input\_, x)

def Decoder():

input\_ = Input(shape=(8, 8, 512))

x = input\_

x = upscale(256)(x)

x = upscale(128)(x)

x = upscale(64)(x)

x = Conv2D(3, kernel\_size=5, padding='same', activation='sigmoid')(x)

return Model(input\_, x)

根据人脸互换所需要的自编码器结构，创建Encoder，Decoder\_A和Encoder，Decoder\_B结构，并且选择绝对平方损失作为模型的损失函数。

from tensorflow.keras.optimizers import Adam

#定义优化器

optimizer = Adam(lr=5e-5, beta\_1=0.5, beta\_2=0.999)

encoder = Encoder()

decoder\_A = Decoder()

decoder\_B = Decoder()

#定义输入函数大小

x = Input(shape=IMAGE\_SHAPE)

#定义解析A类图片的神经网络

autoencoder\_A = Model(x, decoder\_A(encoder(x)))

#定义解析B类图片的神经网络

autoencoder\_B = Model(x, decoder\_B(encoder(x)))

#使用同一个优化器，计算损失和的最小值，损失函数采用平均绝对误差

autoencoder\_A.compile(optimizer=optimizer, loss='mean\_absolute\_error')

autoencoder\_B.compile(optimizer=optimizer, loss='mean\_absolute\_error')

#输出两个对象

autoencoder\_A, autoencoder\_B

20.3.3 训练模型

相关代码如下：

#保存模型

def save\_model\_weights():

encoder .save\_weights("encoder.h5")

decoder\_A.save\_weights("decoder\_A.h5")

decoder\_B.save\_weights("decoder\_B.h5")

print("save model weights")

#开始训练

epochs = 8000

for epoch in range(epochs):

print("第{}代，开始训练。。。".format(epoch))

batch\_size = 26

warped\_A, target\_A = get\_training\_data(images\_A, batch\_size)

warped\_B, target\_B = get\_training\_data(images\_B, batch\_size)

loss\_A = autoencoder\_A.train\_on\_batch(warped\_A, target\_A)

loss\_B = autoencoder\_B.train\_on\_batch(warped\_B, target\_B)

print("lossA:{},lossB:{}".format(loss\_A, loss\_B))

#下面是画图和保存模型的操作

save\_model\_weights()

20.3.4 测试模型

相关代码如下：

#测试的代码和训练代码类似，只是删去了循环和训练的步骤

print("开始加载模型，请耐心等待……")

encoder .load\_weights("encoder.h5")

decoder\_A.load\_weights("decoder\_A.h5")

decoder\_B.load\_weights("decoder\_B.h5")

#下面代码和训练代码类似

#获取图片，并对图片进行预处理

images\_A = get\_image\_paths("trump")

images\_B = get\_image\_paths("cage")

#图片进行归一化处理

images\_A = load\_images(images\_A) / 255.0

images\_B = load\_images(images\_B) / 255.0

images\_A += images\_B.mean(axis=(0, 1, 2)) - images\_A.mean(axis=(0, 1, 2))

batch\_size = 64

warped\_A, target\_A = get\_training\_data(images\_A, batch\_size)

warped\_B, target\_B = get\_training\_data(images\_B, batch\_size)

#分别取当下批次的川普和凯奇图片的前三张进行观察

test\_A = target\_A[0:3]

test\_B = target\_B[0:3]

print("开始预测，请耐心等待……")

#进行拼接原图

figure\_A = np.stack([

test\_A,

autoencoder\_A.predict(test\_A),

autoencoder\_B.predict(test\_A),

], axis=1)

#进行拼接

figure\_B = np.stack([

test\_B,

autoencoder\_B.predict(test\_B),

autoencoder\_A.predict(test\_B),

], axis=1)

print("开始画图，请耐心等待……")

#将多幅图拼成一幅图

figure = np.concatenate([figure\_A, figure\_B], axis=0)

figure = figure.reshape((2, 3) + figure.shape[1:])

figure = stack\_images(figure)

#将图片进行反归一化

figure = np.clip(figure \* 255, 0, 255).astype('uint8')

#显示图片

plt.imshow(cv2.cvtColor(figure, cv2.COLOR\_BGR2RGB))

plt.show()