# 文档特征提取

方法：常用one-hot和TF-IDF

实现：

**sklearn实现one hot encode：**

from sklearn import preprocessing

enc = preprocessing.OneHotEncoder() # 创建对象

enc.fit([[0,0,3],[1,1,0],[0,2,1],[1,0,2]]) # 拟合

array = enc.transform([[0,1,3]]).toarray() # 转化

print(array)

**sklearn实现tfidf：**

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.feature\_extraction.text import TfidfTransformer

tag\_list = ['青年 吃货 唱歌',

'少年 游戏 叛逆',

'少年 吃货 足球']

vectorizer = CountVectorizer() #将文本中的词语转换为词频矩阵

X = vectorizer.fit\_transform(tag\_list) #计算个词语出现的次数

"""

word\_dict = vectorizer.vocabulary\_

{'唱歌': 2, '吃货': 1, '青年': 6, '足球': 5, '叛逆': 0, '少年': 3, '游戏': 4}

"""

transformer = TfidfTransformer()

tfidf = transformer.fit\_transform(X) #将词频矩阵X统计成TF-IDF值

print(tfidf.toarray())

# 图像特征提取

方法：常用LBP特征提取算法、HOG特征提取算法、Haar特征提取算法

**实现：**

**LBP算法实现**

**Language: C++(OpenCV)**

**#include "opencv2/core/core.hpp"**

**#include "opencv2/contrib/contrib.hpp"**

**#include "opencv2/highgui/highgui.hpp"**

**#include <iostream>**

**#include <fstream>**

**#include <sstream>**

**using namespace cv;**

**using namespace std;**

**void elbp(Mat& src, Mat &dst, int radius, int neighbors)**

**{**

**for(int n=0; n<neighbors; n++)**

**{**

**// 采样点的计算**

**float x = static\_cast<float>(-radius \* sin(2.0\*CV\_PI\*n/static\_cast<float>(neighbors)));**

**float y = static\_cast<float>(radius \* cos(2.0\*CV\_PI\*n/static\_cast<float>(neighbors)));**

**// 上取整和下取整的值**

**int fx = static\_cast<int>(floor(x));**

**int fy = static\_cast<int>(floor(y));**

**int cx = static\_cast<int>(ceil(x));**

**int cy = static\_cast<int>(ceil(y));**

**// 小数部分**

**float ty = y - fy;**

**float tx = x - fx;**

**// 设置插值权重**

**float w1 = (1 - tx) \* (1 - ty);**

**float w2 = tx \* (1 - ty);**

**float w3 = (1 - tx) \* ty;**

**float w4 = tx \* ty;**

**// 循环处理图像数据**

**for(int i=radius; i < src.rows-radius;i++)**

**{**

**for(int j=radius;j < src.cols-radius;j++)**

**{**

**// 计算插值**

**float t = static\_cast<float>(w1\*src.at<uchar>(i+fy,j+fx) + w2\*src.at<uchar>(i+fy,j+cx) + w3\*src.at<uchar>(i+cy,j+fx) + w4\*src.at<uchar>(i+cy,j+cx));**

**// 进行编码**

**dst.at<uchar>(i-radius,j-radius) += ((t > src.at<uchar>(i,j)) || (std::abs(t-src.at<uchar>(i,j)) < std::numeric\_limits<float>::epsilon())) << n;**

**}**

**}**

**}**

**}**

**void elbp1(Mat& src, Mat &dst)**

**{**

**// 循环处理图像数据**

**for(int i=1; i < src.rows-1;i++)**

**{**

**for(int j=1;j < src.cols-1;j++)**

**{**

**uchar tt = 0;**

**int tt1 = 0;**

**uchar u = src.at<uchar>(i,j);**

**if(src.at<uchar>(i-1,j-1)>u) { tt += 1 <<tt1; }**

**tt1++;**

**if(src.at<uchar>(i-1,j)>u) { tt += 1 <<tt1; }**

**tt1++;**

**if(src.at<uchar>(i-1,j+1)>u) { tt += 1 <<tt1; }**

**tt1++;**

**if(src.at<uchar>(i,j+1)>u) { tt += 1 <<tt1; }**

**tt1++;**

**if(src.at<uchar>(i+1,j+1)>u) { tt += 1 <<tt1; }**

**tt1++;**

**if(src.at<uchar>(i+1,j)>u) { tt += 1 <<tt1; }**

**tt1++;**

**if(src.at<uchar>(i+1,j-1)>u) { tt += 1 <<tt1; }**

**tt1++;**

**if(src.at<uchar>(i-1,j)>u) { tt += 1 <<tt1; }**

**tt1++;**

**dst.at<uchar>(i-1,j-1) = tt;// 更正，之前是dst.at<uchar>(i,j)=tt;**

**}**

**}**

**}**

**int main()**

**{**

**Mat img = cv::imread("bear.jpg", 0);**

**namedWindow("image");**

**imshow("image", img);**

**int radius, neighbors;**

**radius = 1;**

**neighbors = 8;**

**//创建一个LBP图谱**

**Mat dst = Mat(img.rows-2\*radius, img.cols-2\*radius,CV\_8UC1, Scalar(0));**

**elbp1(img,dst);**

**namedWindow("normal");**

**imshow("normal", dst);**

**Mat dst1 = Mat(img.rows-2\*radius, img.cols-2\*radius,CV\_8UC1, Scalar(0));**

**elbp(img,dst1,1,8);**

**namedWindow("circle");**

**imshow("circle", dst1);**

**cv::waitKey(0);**

**}**

**HOG算法实现**

**Language: C++(OpenCV)**

**#include <iostream>**

**#include <opencv2/opencv.hpp>**

**#include <features2d/features2d.hpp>**

**#include <opencv2/objdetect/objdetect.hpp>**

**#include "opencv2/imgproc/imgproc.hpp"**

**using namespace std;**

**using namespace cv;**

**int main(int argc, char\*\* argv)**

**{**

**Mat image = imread("pep.jpg");**

**if (image.empty())**

**{**

**std::cout << "read image failed" << std::endl;**

**return 0;**

**}**

**namedWindow("src");**

**imshow("src",image);**

**waitKey(0);**

**// 1. 定义HOG对象**

**cv::HOGDescriptor hog;**

**// 2. 设置SVM分类器**

**hog.setSVMDetector(cv::HOGDescriptor::getDefaultPeopleDetector());**

**std::vector<cv::Rect> regions;**

**hog.detectMultiScale(image, regions, 0, cv::Size(8, 8), cv::Size(32, 32), 1.05, 1);**

**// 显示**

**for (size\_t i = 0; i < regions.size(); i++)**

**{**

**cv::rectangle(image, regions[i], cv::Scalar(0, 0, 255), 2);**

**}**

**cv::imshow("hog", image);**

**cv::waitKey(0);**

**return 0;**

**}**

# 视频特征提取

**Python+OpenCv实现视频流的任意局部特征的提取和保存：**

import cv2

import numpy as np

coor\_x,coor\_y = -1, -1 # 初始值并无意义，只是为了能够使用np.row\_stack函数

"""定义视频编码器

FourCC全称Four-Character Codes，代表四字符代码 (four character code),

它是一个32位的标示符，其实就是typedef unsigned int FOURCC;

是一种独立标示视频数据流格式的四字符代码。

因此cv2.VideoWriter\_fourcc()函数的作用是输入四个字符代码即可得到对应的视频编码器。

"""

fourcc = cv2.VideoWriter\_fourcc(\*'XVID') # 使用XVID编码器

camera = cv2.VideoCapture('细胞极体拨动合成视频.wmv') # 从文件读取视频,Todo:只需要修改成自己的视频路径即可进行测试

fps = camera.get(cv2.CAP\_PROP\_FPS)# 获取视频帧率

print('视频帧率：%d fps' %fps)

# 判断视频是否成功打开

if (camera.isOpened()):

print('视频已打开')

else:

print('视频打开失败!')

# # 测试用,查看视频size

size = (int(camera.get(cv2.CAP\_PROP\_FRAME\_WIDTH)),

int(camera.get(cv2.CAP\_PROP\_FRAME\_HEIGHT)))

print ('视频尺寸:'+repr(size))

coor = np.array([[1,1]]) # Todo:初始值并无意义，只是为了能够使用np.row\_stack函数

def OnMouseAction(event,x,y,flags,param):

global coor\_x,coor\_y,coor

if event == cv2.EVENT\_LBUTTONDOWN:

print("左键点击")

print("%s" %x,y)

coor\_x ,coor\_y = x ,y

coor\_m = [coor\_x,coor\_y]

coor = np.row\_stack((coor,coor\_m))

elif event==cv2.EVENT\_LBUTTONUP:

cv2.line(img, (coor\_x, coor\_y), (coor\_x, coor\_y), (255, 255, 0), 7)

elif event==cv2.EVENT\_RBUTTONDOWN :

print("右键点击")

elif flags==cv2.EVENT\_FLAG\_LBUTTON:

print("左鍵拖曳")

elif event==cv2.EVENT\_MBUTTONDOWN :

print("中键点击")

'''

创建回调函数的函数setMouseCallback()；

下面把回调函数与OpenCV窗口绑定在一起

'''

grabbed, img = camera.read() # 逐帧采集视频流

cv2.namedWindow('Image')

cv2.setMouseCallback('Image',OnMouseAction)

while(1):

cv2.imshow('Image',img)

k=cv2.waitKey(1) & 0xFF

if k==ord(' '): # 空格退出操作

break

cv2.destroyAllWindows() # 关闭页面

Width\_choose = coor[2,0]-coor[1,0] # 选中区域的宽

Height\_choose = coor[2, 1] - coor[1, 1] # 选中区域的高

print("视频选中区域的宽：%d" %Width\_choose,'\n'"视频选中区域的高：%d" %Height\_choose)

out = cv2.VideoWriter('output\_test1.avi',fourcc, fps, (Width\_choose,Height\_choose)) # 参数分别是：保存的文件名、编码器、帧率、视频宽高

Video\_choose = np.zeros((Width\_choose, Height\_choose, 3), np.uint8)

while True:

grabbed, frame = camera.read() # 逐帧采集视频流

if not grabbed:

break

gray\_lwpCV = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY) # 转灰度图

frame\_data = np.array(gray\_lwpCV) # 每一帧循环存入数组

box\_data = frame\_data[coor[1,1]:coor[2,1], coor[1,0]:coor[2,0]] # 取矩形目标区域

pixel\_sum = np.sum(box\_data, axis=1) # 行求和q

x = range(Height\_choose)

emptyImage = np.zeros((Width\_choose \* 10, Height\_choose \* 2, 3), np.uint8)

Video\_choose = frame[coor[1,1]:coor[2,1],coor[1,0]:coor[2,0]]

out.write(Video\_choose)

cv2.imshow('Video\_choose', Video\_choose)

for i in x:

cv2.rectangle(emptyImage, (i\*2, (Width\_choose-pixel\_sum[i]//255)\*10), ((i+1)\*2, Width\_choose\*10), (255, 0, 0), 1)

emptyImage = cv2.resize(emptyImage, (320, 240))

lwpCV\_box = cv2.rectangle(frame, tuple(coor[1,:]), tuple(coor[2,:]), (0, 255, 0), 2)

cv2.imshow('lwpCVWindow', frame) # 显示采集到的视频流

cv2.imshow('sum', emptyImage) # 显示画出的条形图

key = cv2.waitKey(1) & 0xFF

if key == ord('q'):

break

out.release()

camera.release()

cv2.destroyAllWindows()

# 参考文献：

https://blog.csdn.net/xiongchao99/article/details/78776629

https://blog.csdn.net/SoaringLee\_fighting/article/details/52699381

https://blog.csdn.net/soaringlee\_fighting/article/details/52693843

https://blog.csdn.net/qq\_30622831/java/article/details/81878067

https://www.cnblogs.com/lianyingteng/p/7755545.html