机器学习与模式识别课程报告

题目：应用 KNN 算法识别手机传感器数据交通方式

组　　员：王倩妮（2015112956）

杜铭枢（2015112082）

陈燕铭（2015110067）

任课教师：郝 莉

2018 年 3 月 19 日

一、问题描述

训练数据文件夹中是已标定好的出行个体在出行过程中的时间（s）、经纬度（度）、速度（m/s）和对应交通方式（1-步行，2-自行车，3-公交车，4-小汽车）。

利用 KNN 算法选取合适特征输入，并计算分类准确率。取得理想结果后，对测试文件夹数据对应的交通出行方式进行预测。

二、数据准备

2.1 数据来源

本数据为交通运输与物流学院杨飞老师课题组开发的手机 APP“行易”采集的不同交通出行方式条件下的手机传感器数据。

2.2 数据描述

观察原始数据，原始数据分为训练数据集与测试数据集。

- ✓ 训练数据集包含共 6 个出行个体、17130 条数据的个体编号、时间、经度、纬度、速度与经人工标定的交通方式。
- ✓ 测试数据包含共 44 个出行个体的个体编号、时间、经度、纬度、速度数据。

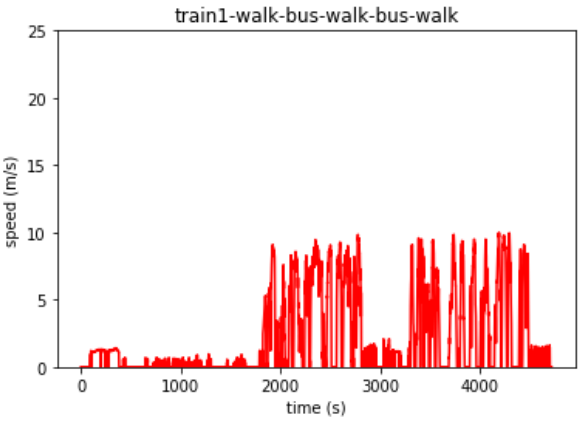
其中交通方式的对应规则为：1-步行，2-自行车，3-公交车，4-小汽车。数据采集频率为 1 次/s。

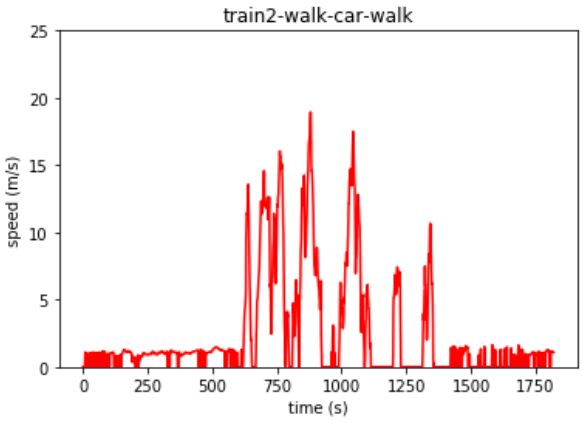
2.3 数据预处理与数据可视化

由于中文在数据处理上的不便性，因此，将以上数据项标题转化为“id_number”、“time”、“longitude”、“latitude”、“speed”、“means”，并用于接下来模型的训练与预测。

经验证原始数据中未存在数据缺失现象，因此，无需进行数据补全工作。

绘制 6 个训练样本的速度随记录数目的变化图像，如下图所示。在其中发现规律，并为模型输入输出的确定提供依据。





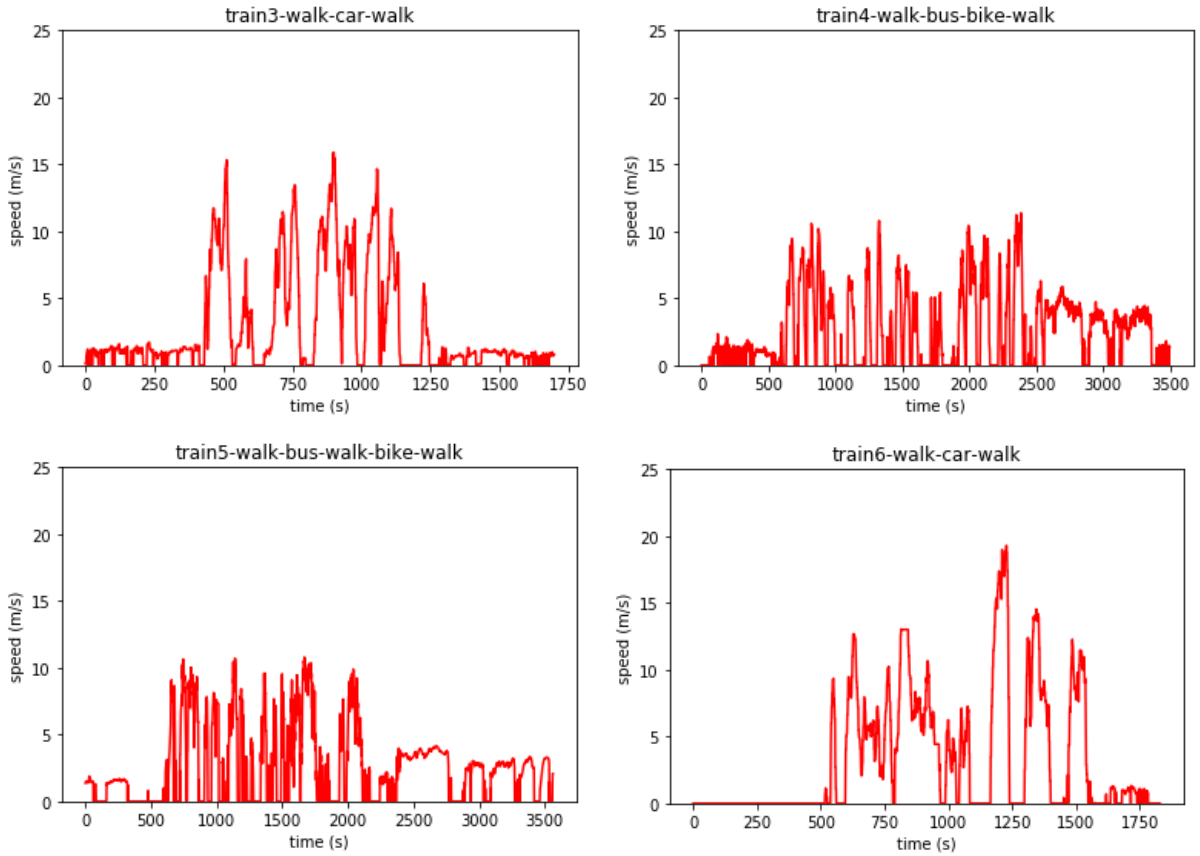


图 1 训练数据集速度变化散点图

从上图中不难发现数据存在以下特征：

1) 步行速度平稳且处于较低水平
2) 自行车速度高于步行速度
3) 公交车与小汽车速度较高
4) 公交车与小汽车的波动性存在差异

因此，我们可以产生以下两条初步结论：

1) 交通方式在短时间内不会频繁变化（不存在频繁跳动）
2) 交通方式与速度、速度变化有关

考虑以上因素，由于数据前后相关性的存在，所以不能直接输入原始数据，而是应该在考虑相关性的基础上处理速度数据。而数据处理的基本思想为：分组思想。在这种思想的指导下，我们对原始数据进行处理，得到 KNN 算法过程中使用的特征。

三、建模与预测——利用 KNN 算法识别交通方式

3.1 KNN 算法搭建环境

- 分类器整体编程语言：Python
- 矩阵运算：Numpy
- 绘图：Matplotlib

- 数据处理：Pandas
- 其他模块：operator, collections

3.2 KNN 算法的特征选取与集合划分

3.2.1 特征量选取

如前文分析所述，数据集中经度、维度等均与交通出行方式无关，因此不选择作为特征量。又由于数据前后存在一定关联性，利用分组“滑动”思想，在 KNN 算法中，选择滚动长度 SCALE=60，即每个数据的前 60 个数据与后 60 个数据进行运算，每个出行个体的前 SCALE 与后 SCALE 个数据分别处理。并增加了数据输入项。

每条原始数据经处理，拥有如下 5 个特征作为 KNN 分类计算距离的依据：

- 原始速度
- 原始速度时间序列下的一次指数平滑值，阻尼=0.5
- 滚动每 2SCALE 内的平均速度 $\bar{v}$
- 滚动每 2SCALE 内的最大速度 v_{max}
- 滚动每 2SCALE 内的速度标准差 $std(v)$

由于 KNN 算法无需将输出转为 one-hot 的形式，因此分类标签为：

- 原始标定的出行方式（以数字代替）[1],[2],[3],[4]

6 个出行个体的全部 5 项数据汇总为 data_x，所有标定的出行方式汇总为 data_y。

3.2.2 训练集与验证集

在以上 6 个出行个体构成的全部样本中，随机选取 80%作为训练数据，这些训练数据用于构建 KNN 算法，作为距离计算的基础。剩余 20%为验证数据，将这部分数据利用 KNN 算法确定的分类结果与标定的数据标签进行比较，计算错误率。由于交通出行方式具有一定程度的连续型，因此随机选取训练集与验证集，比直接分段指定计算得出的错误率更为科学、合理。

3.3 KNN 算法的工作原理与参数构建

3.3.1 KNN 算法原理

在训练样本集中每个数据都存在标签，即我们知道样本集中每一数据与所属分类的对应关系。输人没有标签的新数据后，将新数据的每个特征与样本集中数据对应的特征进行比较，然后算法提取样本集中特征最相似数据（最近邻）的分类标签。选择 k 个最相似数据中出现次数最多的分类，作为新数据的分类。在 KNN 中，通过计算对象间距离作为各个对象之间的相似性指标，代替对象之间的匹配度计算。

对于训练样本数为 m，特征数为 n 的训练样本集，计算测试样本 x 与 m 个训练样本的欧氏距离

$$d(x, y_i) = \sqrt{\sum_{j=1}^{n} (x_j - y_j)^2}$$

其中 i=1，2，……，m。对 $d(x, y_i)$ 进行降序排列，选择前 k 个值，出现次数最多的分类作为测试样本 x 的分类。

3.3.2 KNN 分类器模型构建

完成特征量的选取、计算，进行训练集与验证集的划分后，主要定义了用于实现 KNN 分类的 classify0 函数，用于进行标准化的 autoNorm 函数与用于进行交通方式划分的 TrafficClassTest 函数。

现分别对以下三个函数进行介绍。

函数 1：classify0 函数

```
def classify0(inX, dataSet, labels, k):
    dataSetSize = dataSet.shape[0]
    diffMat = np.tile(inX, (dataSetSize,1)) - dataSet
    sqDiffMat = diffMat**2
    sqDistances = sqDiffMat.sum(axis=1)
    distances = sqDistances**0.5
    sortedDistIndicies = distances.argsort()
    #print(sortedDistIndicies)
    #print(len(sortedDistIndicies))
    classCount={}
    for i in range(k):
        voteIlabel = labels[sortedDistIndicies[i]]
        voteIlabel=voteIlabel[0]####这一步不做就会报错
        classCount[voteIlabel] = classCount.get(voteIlabel,0) + 1
        #print('step:',i,' voteIlabel:',voteIlabel)
        #print(classCount)
    sortedClassCount = sorted(classCount.items(), key=operator.itemgetter(1), reverse=True)
    #print('result:',sortedClassCount[0][0])
    return sortedClassCount[0][0]
```

函数输入 inX 为验证/测试集的每条需进行分类的数据，dataSet 为计算欧氏距离的基准数据，即全部训练集，labels 为训练集的标签，k 为最近邻算法选取的“邻居”数。函数首先获取训练集样本数，计算输入数据与训练集数据的欧式距离，并对距离进行排序，得到与输入数据距离最近的 k 个训练集样本，并计算、返回 k 个样本中占多数的类别标签作为输入数据的预测分类标签。

函数 2：autoNorm 函数

```
def autoNorm(dataSet):
    minVals = dataSet.min(axis=0)
    maxVals = dataSet.max(axis=0)
```

```
    ranges = maxVals - minVals
    normDataSet = np.zeros(np.shape(dataSet))
    m = dataSet.shape[0]
    normDataSet = dataSet - np.tile(minVals, (m,1))
    normDataSet = normDataSet/np.tile(ranges, (m,1))      #element wise divide
    return normDataSet, ranges, minVals
```

autoNorm 函数完成对于特征值的标准化，标准化后可以避免特征值间极值的差异性对分类结果造成的影响。函数首先计算“列”方向的数据最小值、最大值，得到数据的范围，再利用标准化公式对于数据进行标准化，最终返回标准化的特征值、数据范围与数据最小值。在函数中应用 np.tile 命令进行矩阵的复制，使矩阵的维度相同从而能够进行运算。标准化公式为：

$$\text{norm} = \frac{actual - min}{max - min}$$

函数 3：TrafficClassTest 函数

```
def TrafficClassTest():
    normMat, ranges, minVals = autoNorm(data_xnew)
    normMat1, ranges1, minVals1 = autoNorm(Data_xnew)
    numTestVecs = Data_ynew.shape[0]
    errorCount = 0.0
    for i in range(numTestVecs):
        classifierResult = classify0(normMat1[i,:],normMat,data_ynew,3)
        print ("the classifier came back with: %d, the real answer is: %d" %
(classifierResult,Data_ynew[i]))
        if (classifierResult != Data_ynew[i]): errorCount += 1.0
    print ("the total error rate is: %f" % (errorCount/float(numTestVecs)))
```

TrafficClassTest 函数用于完成整个交通方式分类器的流程。首先，对于训练集与验证集的特征值进行标准化，即调用 autoNorm 函数；其次得到验证集的样本数，初始化错误数计数器，而后循环遍历验证集每一条样本，利用 KNN 算法（即调用 classify0 函数）确定该样本交通方式的分类，并与实际人工标定结果进行比较，若结果错误，则错误数增加 1，最终计算得到整个分类器的错误率。

四、模型的结果、评估与优化

4.1 模型结果

4.1.1 错误率定义

利用随机选取的 20%数据计算错误率，比较利用 KNN 算法分类所得标签与人工标定标签结果是否一致，若不一致，则错误样本数加一。最终利用错误样本数除以总验证样本数得到错误率。

$$\text{error rate} = \frac{errorCount}{validCount}$$

若错误率越低，则说明算法应用后分类效果越好。但由于训练样本与验证样本选取的随机性，因此，每次程序运行结果会存在一定差异性。

4.1.2 运行结果

由于训练集与验证集为按比例随机选取，因此每次的程序运行结果存在一定差异性，多次运行程序后，发现错误率在 0.02-0.04 之间。下一步，可分方式计算识别率，并发现易混淆的交通方式，提出进一步解决措施。

依据标签将验证数据分为四种交通方式，并分别计算在该标签下预测结果为各类交通方式的概率。该参数计算可以辨认出易混淆的交通方式，看出分类器针对不同交通方式实施效果的优劣。

$$识别率 = \frac{预测数据中该种方式的数据}{标签(真实)数据中该种方式的数目}$$

以某次运行结果为例，运行后显示该次分类器分类错误率为：0.030940(3.094%)，即在 3426 个验证数据中，有 106 个出现了分类错误。分方式计算识别率，结果如下（保留四位小数）：

表 1 识别率分布图

标签 \ 预测	1：步行	2：自行车	3：公交车	4：小汽车	求和
1：步行	0.9767	0.0021	0.0163	0.0049	1.0000
2：自行车	0.0024	0.9904	0.0072	0.0000	1.0000
3：公交车	0.0230	0.0000	0.9641	0.0129	1.0000
4：小汽车	0.0138	0.0000	0.0453	0.9409	1.0000

4.2 模型的评估与优化

由以上结果可知，该模型对于步行与自行车的识别效果较好，公交车与小汽车识别效果一般，且步行最易被识别为公交车；自行车最易被识别为公交车；公交车最易最易被识别为步行；小汽车最易被识别为公交车。但由于原始数据的选择和训练验证集的差异，各次运行结果具有一定差异性，还应当多运行几次进行具体分析。

未来可以对输入特征值、滑动平均阻尼数、滚动分组组长、k 邻近算法的 k 值等参数进行调整，从而进一步优化模型，得到效果更优的分类器。

五、应用算法 KNN 心得体会

KNN 算法是一种简单的分类算法，其具备精度高、容许异常值、无需输入假定的优点，但同时也拥有计算、空间复杂度高的缺陷。对于这类数据量小的数据集，采用 KNN 算法的分类效果良好，甚至超越 SVM、神经网络等分类方法的分类准确率。

在对于所选数据集使用 KNN 算法的过程中，我们也遇到了一些问题，比如：

① Numpy 的 array 与 Python 中的 List、Dictionary、Tuple 等数据容器的操作命令不同，部分条件下不注意会存在错误。

② 对数据特征进行思考，选取合适的特征进行距离计算对于准确率/错误率结果至

关重要。

③ 部分情况下需注意数据的维度问题，很多条件下的错误都是由于在计算过程中维度不对应而产生的。

在实践过程中我们采用了有效手段解决了上述问题，加深了对于 KNN 算法的理解，并通过实战了解了定义函数过程中每个步骤的意义，有了很大的收获。

附录程序源代码

```
#-*- coding=utf-8 -*-
##一次平滑用于预测
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from collections import Counter
import operator


##定义用于将多维 list 整合成为 1 维 list 的函数
def flat(l):
    for k in l:
        if not isinstance(k, (list, tuple)):
            yield k
        else:
            yield from flat(k)


SCALE = 60#定义 SCALE 长度，滚动每段数据实际为 2SCALE 长度
data_x=[]
data_y=[]#训练集

Data_x=[]
Data_y=[]#测试集
##定义文件读入函数
def choose_file(filenumber):
    traindata = pd.read_excel('train%s.xlsx'%filenumber)
    tempspeed=[]
    tempfirstplant=[]
    tempmeans=[]
    tempspeed.append(list(traindata['speed']))
    tempmeans.append(list(traindata['means']))
    tempfirstplant.append(list(traindata['firstplant']))
    tempspeed = list(flat(tempspeed))
    tempmeans = list(flat(tempmeans))
    tempfirstplant=list(flat(tempfirstplant))
    for i in range(SCALE):#每个文件前 SCALE 个数据的读入
        speed=np.array(tempspeed[0 : i+SCALE])
        speed_in=tempspeed[i]
        firstplant=tempfirstplant[i]
        mean_speed=speed.mean()
        max_speed=speed.max()
        std_speed=speed.std()
        data_x.append(list([speed_in,firstplant,mean_speed,max_speed,std_speed]))
        data_y.append(list([tempmeans[i]]))
    for i in range(SCALE,(len(tempspeed)-SCALE)):#每个文件可以正常选择 SCALE 的数据的读入
        speed = np.array(tempspeed[i-SCALE : i+SCALE])
        speed_in=tempspeed[i]
        firstplant=tempfirstplant[i]
        mean_speed=speed.mean()
        max_speed=speed.max()
        std_speed=speed.std()
        data_x.append(list([speed_in,firstplant,mean_speed,max_speed,std_speed]))
        data_y.append(list([tempmeans[i]]))
    for i in range((len(tempspeed)-SCALE),len(tempspeed)):#每个文件尾部的 SCALE 个数据的读入
```

```
            speed=np.array(tempspeed[i-SCALE:(len(tempspeed))])
            speed_in=tempspeed[i]
            firstplant=tempfirstplant[i]
            mean_speed=speed.mean()
            max_speed=speed.max()
            std_speed=speed.std()
            data_x.append(list([speed_in,firstplant,mean_speed,max_speed,std_speed]))
            data_y.append(list([tempmeans[i]]))
    return data_x,data_y#返回用于训练模型的 x 和 y

##读入 5 个出行个体的训练文件
l=[1,2,3,4,5,6]
for i in l:
    choose_file(i)
data_y=np.array(data_y)

import  random
a=range((len(data_x)))
b=random.sample(a,int(0.8*len(data_x)))
c=list(set(a).difference(set(b)))
data_xnew=[]
data_ynew=[]
Data_xnew=[]
Data_ynew=[]
for i in b:
            data_xnew.append(data_x[i])
            data_ynew.append(data_y[i])
for i in c:
            Data_xnew.append(data_x[i])
            Data_ynew.append(data_y[i])


data_xnew=np.array(data_xnew)
data_ynew=np.array(data_ynew)
Data_xnew=np.array(Data_xnew)
Data_ynew=np.array(Data_ynew)


def classify0(inX, dataSet, labels, k):
    dataSetSize = dataSet.shape[0]
    diffMat = np.tile(inX, (dataSetSize,1)) - dataSet
    sqDiffMat = diffMat**2
    sqDistances = sqDiffMat.sum(axis=1)
    distances = sqDistances**0.5
    sortedDistIndicies = distances.argsort()
    #print(sortedDistIndicies)
    #print(len(sortedDistIndicies))
    classCount={}
    for i in range(k):
        voteIlabel = labels[sortedDistIndicies[i]]
        voteIlabel=voteIlabel[0]####这一步不做就会报错
        classCount[voteIlabel] = classCount.get(voteIlabel,0) + 1
        #print('step:',i,' voteIlabel:',voteIlabel)
        #print(classCount)
    sortedClassCount = sorted(classCount.items(), key=operator.itemgetter(1), reverse=True)
    #print('result:',sortedClassCount[0][0])
    return sortedClassCount[0][0]
```

```
def autoNorm(dataSet):
    minVals = dataSet.min(axis=0)
    maxVals = dataSet.max(axis=0)
    ranges = maxVals - minVals
    normDataSet = np.zeros(np.shape(dataSet))
    m = dataSet.shape[0]
    normDataSet = dataSet - np.tile(minVals, (m,1))
    normDataSet = normDataSet/np.tile(ranges, (m,1))      #element wise divide
    return normDataSet, ranges, minVals
CLASSFIERRESULT=[]
def TrafficClassTest():
    normMat, ranges, minVals = autoNorm(data_xnew)
    normMat1, ranges1, minVals1 = autoNorm(Data_xnew)
    numTestVecs = Data_ynew.shape[0]
    errorCount = 0.0
    for i in range(numTestVecs):
        classifierResult = classify0(normMat1[i,:],normMat,data_ynew,3)
        CLASSFIERRESULT.append(classifierResult)
        #print ("the classifier came back with: %d, the real answer is: %d" % (classifierResult,
Data_ynew[i]))
        if (classifierResult != Data_ynew[i]): errorCount += 1.0
    print ("the total error rate is: %f" % (errorCount/float(numTestVecs)))
    print(errorCount,numTestVecs)
TrafficClassTest()
CLASSFIERRESULT = np.array(CLASSFIERRESULT)
#Data_ynew 标签（真实）true；CLASSFIERRESULT 预测 pre
count1,count2,count3,count4=0,0,0,0
true_walk = Data_ynew[Data_ynew==1].shape[0]
true_walk_index = np.argwhere(Data_ynew==1)[:,0]
for o in true_walk_index:
    if CLASSFIERRESULT[o]==1:
        count1+=1
    elif CLASSFIERRESULT[o]==2:
        count2+=1
    elif CLASSFIERRESULT[o]==3:
        count3+=1
    elif CLASSFIERRESULT[o]==4:
        count4+=1
    else:pass
w_w,w_b,w_s,w_c=count1/true_walk,count2/true_walk,count3/true_walk,count4/true_walk


count1,count2,count3,count4=0,0,0,0
true_bike = Data_ynew[Data_ynew==2].shape[0]
true_bike_index = np.argwhere(Data_ynew==2)[:,0]
for o in true_bike_index:
    if CLASSFIERRESULT[o]==1:
        count1+=1
    elif CLASSFIERRESULT[o]==2:
        count2+=1
    elif CLASSFIERRESULT[o]==3:
        count3+=1
    elif CLASSFIERRESULT[o]==4:
        count4+=1
    else:pass
b_w,b_b,b_s,b_c=count1/true_bike,count2/true_bike,count3/true_bike,count4/true_bike
```

```
count1,count2,count3,count4=0,0,0,0
true_bus = Data_ynew[Data_ynew==3].shape[0]
true_bus_index = np.argwhere(Data_ynew==3)[:,0]
for o in true_bus_index:
    if CLASSFIERRESULT[o]==1:
        count1+=1
    elif CLASSFIERRESULT[o]==2:
        count2+=1
    elif CLASSFIERRESULT[o]==3:
        count3+=1
    elif CLASSFIERRESULT[o]==4:
        count4+=1
    else:pass
s_w,s_b,s_s,s_c=count1/true_bus,count2/true_bus,count3/true_bus,count4/true_bus


count1,count2,count3,count4=0,0,0,0
true_car = Data_ynew[Data_ynew==4].shape[0]
true_car_index = np.argwhere(Data_ynew==4)[:,0]
for o in true_car_index:
    if CLASSFIERRESULT[o]==1:
        count1+=1
    elif CLASSFIERRESULT[o]==2:
        count2+=1
    elif CLASSFIERRESULT[o]==3:
        count3+=1
    elif CLASSFIERRESULT[o]==4:
        count4+=1
    else:pass
c_w,c_b,c_s,c_c=count1/true_car,count2/true_car,count3/true_car,count4/true_car
print('line1',w_w,w_b,w_s,w_c)
print('line2',b_w,b_b,b_s,b_c)
print('line3',s_w,s_b,s_s,s_c)
print('line4',c_w,c_b,c_s,c_c)
```