实验题目:

基于 k-NN 算法的 Car Evaluation 的设计与实现

目录

摘要

本次项目共包含 3 个文件，分别是数据集 car.data，数据预处理程序 dataPreprocess.py，分类器 k-NN 算法实现与分类器测试程序 carEvalutionWithKNN.py。本项目程序包含功能为：数据集读写、自然语言转数字语言、数据集随机打乱、数据归一化、异常数据处理、k-NN 算法实现、混淆矩阵的实现、评估指标的计算等。其中，除了 k-NN 算法的实现参照网上的实现代码外，其余代码均为自己实现。经测试模型正常，各个评估指标计算正常。

本实验报告共分为 8 章，第一章，环境搭建。第二章包含本数据集现有的研究成果与进展。第三章，数据预处理，主要包括实验数据分析、预处理、相关代码实现、预处理后数据展示。第四章，分类器算法的实现，包括 k-NN 算法思想简述与代码实现。第五章，分类器评估指标的实现，主要包括混淆矩阵、accuracy、precision、recall、F1 的计算与代码实现。第六章，分类器的测试，主要包含分类器的运行测试与结果展示。第七章是实验总结，第八章是参考资料。

一、环境搭建

实验软件环境：Python、numpy、pandas
实验硬件要求：不限

二、实验题目分析与文献查找

2.1 相关资料与文献查找

经资料查找与文献搜寻发现本实验是数据挖掘（Data Mining）或者机器学习（Machine Learning）课程中一个十分经典的分类问题（Classification Problem），并且是一个典型的多分类问题，有很多研究者根据此数据做过案例研究（Case Study）。

研究者们使用的分类器包括：

- ArtificialNeural Network
- Naive Bayesian
- 10-Folds Cross Validation
- Random Forest
- Decision Tree
- k-NearestNeighbor

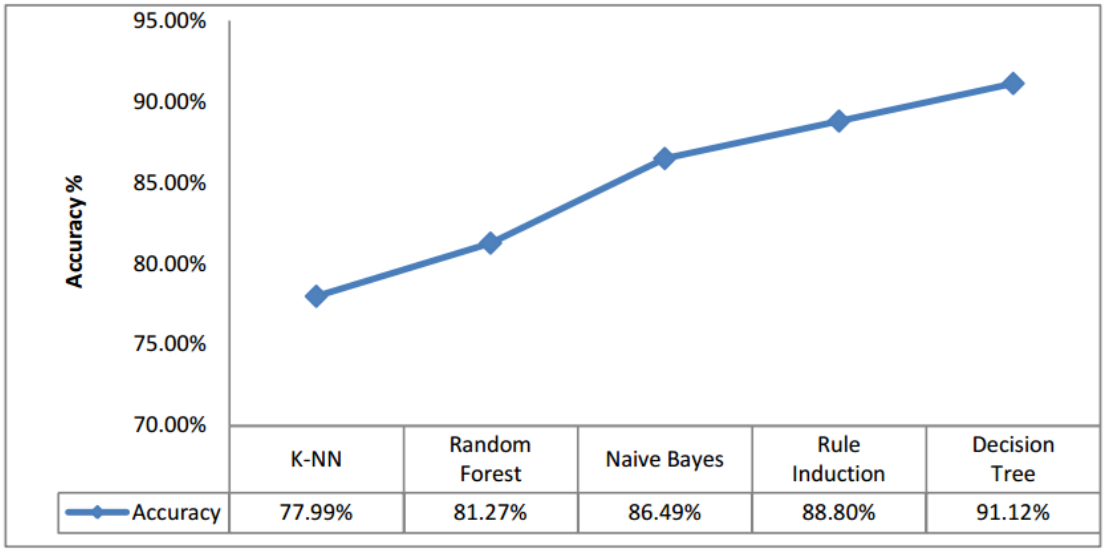
根据《A Case Study on Car Evaluation and Prediction:Comparative Analysis using

Data Mining Models》论文中的实验，可以得到以下初步的概览。

criterion	K-NN(%)	Random forest (%)	Naïve bayes(%)	Rule induction (%)	Decision tree(%)
accuracy	77.9	81.2	86.4	88.8	91.1
Classification error	22.0	18.7	13.5	11.2	8.8
kappa	0.39	0.51	0.68	0.76	0.80
Weighed mean recall	36.7	37.7	65.2	79.1	78.3
Weighed mean precision	83.7	37.5	78.6	83.6	78.0

2.2 本实验所做工作

根据论文的实验结果我们可以直观的看到 k-NN 算法下的精度（accuracy）最低，只有 77.9%；而且决策树算法（Decision Tree）下的精度最高约为 91.1%。当然结果仅限于参考，这个精度与研究者的数据处理方法以及算法的实现有很大的关系。



鉴于 k-NN 的实验结果精度最低，本次实验选用 k-NN 算法进行分类器的设计，目的在于探求上述论文中的实验结果是否合理，以及 k-NN 是不是真的不适合处理本实验的数据。

本文中的数据预处理、k-NN 算法、混淆矩阵、评价指标计算等均为自己实现的，并非使用 TensorFlow 或者 sklearn（scikit-learn）等机器学习库的实现。K-NN 的实现比较简单，距离计算使用的是欧式距离：

$$d(x,y)=\sqrt{\sum_{k=1}^{n}(x_k-y_k)^2}$$

具体代码实现详见本文第四部分，此处不再赘述。

三、数据预处理

3.1 实验数据分析

众所周知，数据挖掘中最关键的前期步骤就是数据预处理，数据预处理的主要内容包括数据清洗、数据集成、数据变换和数据规约。

经分析，本次实验数据格式十分规整，可操作性强，数据值也不存在缺失的情况，如下表所示，数据共计 6 个不同的属性（buying, maint, doors, persons, lug_boot, safety），4 种类别（unacc, acc, good, vgood）。

属性值（attributes）	取值情况
buying	v-high, high, med, low
maint	v-high, high, med, low
doors	2, 3, 4,5more
persons	2,4,more
lug_boot	small, med, med
safety	low, med, high

分类值	取值情况
class values	unacc, acc, good, vgood

仔细观察发现本次数据集共计 1728 条，属性值与分类值同处一行，并且其排列十分有规律，是按照属性值 buying 由高到底（v-high –> high –> med –> low）排列的，这种情况下不利于分类器的训练，要达到最高的训练效果，数据集要尽可能随机一些。

3.2 实验数据预处理工作

根据上述对本次数据集的观察分析，实验数据集的处理过程主要分为以下步骤：

1) 删除缺失数据（空格，NaN，None 等）
2) 自然语言转数字语言
3) 特征、标签分离
4) 数据随机打乱处理

5) 归一化

数据集自然语言转换为数字语言的对应关系如下，

属性	自然语言	处理后数字值
Buying	vhigh	4
	High	3
	Med	2
	Low	1
Maint	vhigh	4
	High	3
	Med	2
	Low	1
Doors	2	2
	3	3
	4	4
	5more	5
Persons	2	2
	4	4
	more	6
Lug_Boot	Small	3
	Med	2
	Big	1
Safety	Low	1
	Med	2
	High	3
Class values	uacc	1
	acc	2
	good	3
	vgood	4

3.3 数据预处理核心代码实现

核心实现代码如下：

```
#数据预处理整个过程
def dataCarMatrix():
    data = loadData() #加载car.data数据
    nomis = removeMissing(data) #删除空格等确实数据
    dataNum = nature2num(nomis) #将自然语言映射为数字
    dataRandom = dataNum.sample(frac=1)  #打乱数据顺序，frac=1是选取全部数据
    allMatrix = dataRandom.as_matrix() #dataframe转化为matrix
    carMatrix = allMatrix[:,0:6] #特征提取，前6列是属性值（特征值）
    lableVector = allMatrix[:,-1] #分类标签label提取，最后一列是label值
    #print(carMatrix) #test
    #print(lableVector) #test
    return carMatrix, lableVector
```

实现自然语言与数字映射的代码：

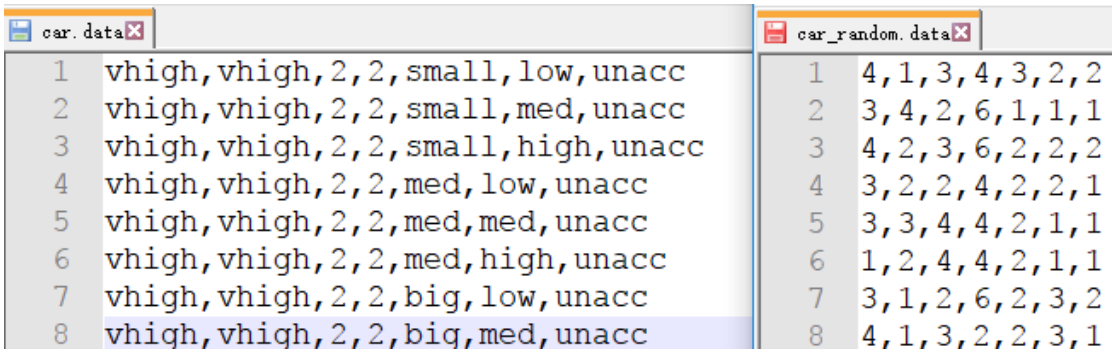
```
# 自然语言映射为数字
def nature2num(df):
    level_mapping = {'low': 1,
                     'med': 2,
                     'high': 3,
                     'vhigh': 4}
    size_mapping = {'small': 1,
                    'med': 2,
                    'big': 3}
    class_mapping = {'unacc': 1,
                     'acc': 2,
                     'good': 3,
                     'vgood': 4}
    num_mapping = {'1': 1,
                   '2': 2,
                   '3': 3,
                   '4': 4,
                   '5more': 5,
                   'more': 6}

    df[0] = df[0].map(level_mapping)
    df[1] = df[1].map(level_mapping)
    df[2] = df[2].map(num_mapping)
    df[3] = df[3].map(num_mapping)
    df[4] = df[4].map(size_mapping)
    df[5] = df[5].map(level_mapping)
    df[6] = df[6].map(class_mapping)
    return df
```

数据归一化代码：

```
#数据归一化
def autoNorm(DataSet):
    minVals = DataSet.min(0 )#将每列中的最小值放在变量minVals中
    maxVals = DataSet.max(0) #获取数据集中每一列的最大数值
    ranges = maxVals - minVals #最大值与最小的差值
    normDataSet = np.zeros(np.shape(DataSet)) #生成一个与dataSet相同的零矩
    m = DataSet.shape[0] #求出dataSet列长度
    normDataSet = DataSet - np.tile(minVals, (m, 1)) #求出oldValue - min
    #把最大最小差值扩充为dataSet同shape，然后作商，是指对应元素进行除法运算
    normDataSet = normDataSet / np.tile(ranges, (m,1))
    return normDataSet, ranges, minVals
```

代码实现过程中仅仅使用了 pandas 和 numpy 两个基础的编程库，未使用其他已经封装好的代码库，数据预处理其余代码详见实验 dataPreprocess.py 代码。

3.4 数据预处理后的数据展示



car.data

```
1 vhigh,vhigh,2,2,small,low,unacc
2 vhigh,vhigh,2,2,small,med,unacc
3 vhigh,vhigh,2,2,small,high,unacc
4 vhigh,vhigh,2,2,med,low,unacc
5 vhigh,vhigh,2,2,med,med,unacc
6 vhigh,vhigh,2,2,med,high,unacc
7 vhigh,vhigh,2,2,big,low,unacc
8 vhigh,vhigh,2,2,big,med,unacc
```

car_random.data

```
1 4,1,3,4,3,2,2
2 3,4,2,6,1,1,1
3 4,2,3,6,2,2,2
4 3,2,2,4,2,2,1
5 3,3,4,4,2,1,1
6 1,2,4,4,2,1,1
7 3,1,2,6,2,3,2
8 4,1,3,2,2,3,1
```

四、分类器算法实现

4.1 k-NN 算法思想

k-NearestNeighbor 算法是有监督分类算法，其算法主体思想就是根据距离相近的邻居类别，来判定自己的所属类别。在训练集中数据和标签已知的情况下，输入测试数据，将测试数据的特征与训练集中对应的特征进行相互比较，找到训练集中与之最为相似的前 K 个数据，则该测试数据对应的类别就是 K 个数据中出现次数最多的那个分类，其算法的描述为：

1）计算测试数据与各个训练数据之间的距离；

2）按照距离的递增关系进行排序；

3）选取距离最小的 K 个点；

4）确定前 K 个点所在类别的出现频率；

5）返回前 K 个点中出现频率最高的类别作为测试数据的预测分类。

本实验中的数据集的属性和标签已知，符合需要有一个已被标记类别的训练数据集的前提，故适用 k-NN 算法。

4.2 k-NN 算法的代码实现

以下是 k-NN 算法简单实现，通过计算欧式距离作为算法思想中的距离的实现。

```
#基于k-NN的分类器的实现
def classifyCarWithKNN(CarData, DataSet, Labels, k):
    DataSetSize = DataSet.shape[0] #获取矩阵第一纬度的长度
    DiffMat = tile(CarData, (DataSetSize, 1)) - DataSet
    sqDiffMat = DiffMat**2
    sqDistances = sqDiffMat**0.5  #计算欧式距离
    distances = sqDistances.sum(axis=1) #矩阵行相加。生成新矩阵
    sortedDistIndicies = distances.argsort() #返回矩阵中的数组从小到大的下标值，返回新矩阵
    classCount = {}  #初始化新字典
    for i in range(k):
        voterLabel = Labels[sortedDistIndicies[i]]
        classCount[voterLabel] = classCount.get(voterLabel, 0) + 1
    sortedClassCount = sorted(classCount.items(), key = operator.itemgetter(1), reverse=True)
    return sortedClassCount[0][0]
```

当然，现在有很多机器学习的库已经都很完整的实现了 k-NN 算法，基本上直接调用就可以，比如 sklearn（scikit-learn）的库已经封装好了完整的 k-NN 算法实现以及分类器的评价指标。但是本次实验还是想自己实现 k-NN 算法，对数据挖掘的过程有一个更深刻的理解。

五、分类器评价指标实现

5.1 多分类问题下的混淆矩阵

在机器学习领域，混淆矩阵（confusion matrix），又称为可能性表格或是错误矩阵。它是一种特定的矩阵用来呈现算法性能的可视化效果，通常是监督学习（非监督学习，通常用匹配矩阵：matching matrix）。其每一列代表预测值，每一行代表的是实际的类别。基本的混淆矩阵是二分类问题的混淆矩阵，如下所示：

ConfusionMatrix		predict	
		1	0
real	1	TP	FN
	0	FP	TN

其中各个值得含义如下：

- TP：真正类(TP)，样本的真实类别是正类，并且模型预测的结果也是正类
- FN：假负类(FN)，样本的真实类别是正类，但是模型将其预测成为负类。
- FP：假正类(FP)，样本的真实类别是负类，但是模型将其预测成为正类。
- TN：真负类(TN)，样本的真实类别是负类，并且模型将其预测成为负类。

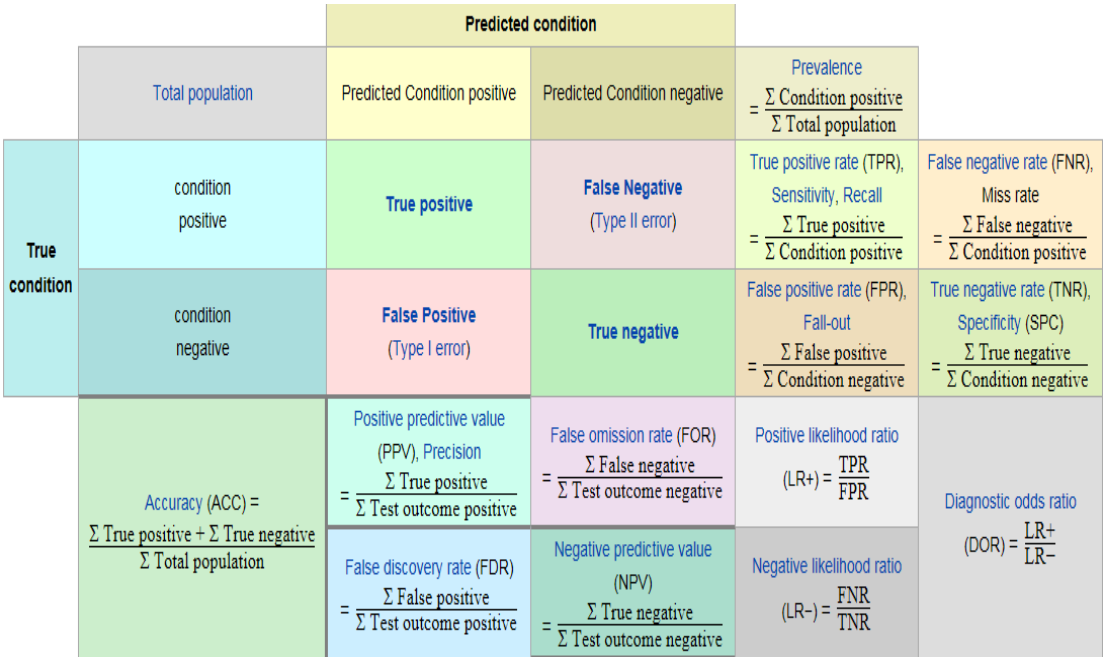
本次实验是多分类问题，对于多分类问题的混淆矩阵，是使用类别（label）值代替二分类问题中的 TRUE 和 FALSE，形成的混淆矩阵如下，多分类混淆矩阵的对角线上的值是预测正确的样本数量。

ConfusionMatrix		predict			
		uacc	acc	good	vgood
real	uacc				
	acc				

	good				
	vgood				

5.2 模型评估指标

根据混淆矩阵可以衍生出各种评价指标，下图是维基百科上的一张经典图，详细列述了各个评价指标与混淆矩阵各个值之间的关系。



■ accuracy 模型的精度，

$$\text{accuracy} = \frac{TP + TN}{TP + FN + FP + TN}$$

即模型预测正确的个数 / 样本的总个数，一般情况下，模型的精度越高，说明模型的效果越好。

■ precision 查准率，阳性预测值。在模型预测为正类的样本中，真正为正类的样本所占的比例。

$$\text{precision} = \frac{TP}{TP + FP}$$

一般情况下，体现了模型对负样本的区分能力，precision 越高，说明模型对负样本的区分能力越强。

■ recall 召回率，真正类率，表示的是，模型预测为正类的样本的数量，占总的正类样本数量的比值。

$$\text{recall} = \frac{TP}{TP + FN}$$

一般情况下，Recall 越高，说明有更多的正类样本被模型预测正确，模型的效果越好。体现了分类模型对正样本的识别能力，

- F1-score

$$\mathrm{F1} = \frac{2TP}{2TP + FN + FP} = \frac{2 \times precision \times recall}{precision + recall}$$

可以看到， F1-score 是 precision 和 recall 两者的综合。F1-score 越高，说明分类模型越稳健。

5.3 混淆矩阵的代码实现

根据上述分析，根据真实值 real 和预测值 predict 的比较可以判断出每次样本测试对应混淆矩阵的哪个元素，比如说如果 real == 1 并且 predict == 1，那么混淆矩阵的第 0 行第 0 列元素就增加 1（cMat[0][0] += 1），以此类推，样本测试完毕就可以获得本次测试的混淆矩阵。

具体代码如下：

```
#计算混淆矩阵
def getConfusionMatrix(cMat,real,predict):
    if(real == 1 and predict == 1):
        cMat[0][0] += 1
    if(real == 1 and predict == 2):
        cMat[0][1] += 1
    if(real == 1 and predict == 3):
        cMat[0][2] += 1
    if(real == 1 and predict == 4):
        cMat[0][3] += 1
    if(real == 2 and predict == 1):
        cMat[1][0] += 1
    if(real == 2 and predict == 2):
        cMat[1][1] += 1
    if(real == 2 and predict == 3):
        cMat[1][2] += 1
    if(real == 2 and predict == 4):
        cMat[1][3] += 1
    if(real == 3 and predict == 1):
        cMat[2][0] += 1
    if(real == 3 and predict == 2):
        cMat[2][1] += 1
    if(real == 3 and predict == 3):
        cMat[2][2] += 1
    if(real == 3 and predict == 4):
        cMat[2][3] += 1
    if(real == 4 and predict == 1):
        cMat[3][0] += 1
    if(real == 4 and predict == 2):
```

```
            cMat[3][1] += 1
        if(real == 4 and predict == 3):
            cMat[3][2] += 1
        if(real == 4 and predict == 4):
            cMat[3][3] += 1
        #print(cMat)
        return cMat
```

5.4 评价指标计算的实现

根据对混淆矩阵以及评价指标的分析，在 5.3 中已经实现了混淆矩阵，那么我们根据 5.2 的计算公式，遍历混淆矩阵可以获得公式中所使用的值。

遍历混淆矩阵对角线的元素，可以获得预测与实际都正确的数量，那么可以计算 accuracy。

分别求出混淆矩阵每行、每列的值，就可以得到 5.2 中 precision 和 recall 的分母，其各自的分子是每类预测正确的值，所以可以得到 precision 和 recall。

F1 的值可以根据 precision 和 recall 套用公式求得。因此，根据混淆矩阵可以求得 accuracy、precision、recall 以及 F1。具体实现代码如下，其中有明确的注释。本次评价指标的计算由自己实现，未使用封装好的库函数。

```
58  #根据混淆矩阵计算accuracy, precision, recall,F1
59  def getEvaluation(confusion_matrix,totalNumOfTestData):
60      accu = [0,0,0,0]
61      column = [0,0,0,0]
62      line = [0,0,0,0]
63      accuracy = 0
64      recall = 0
65      precision = 0
66      for i in range(0,4):
67          accu[i] = confusion_matrix[i][i] #遍历对角线，获得预测正确的值
68      for i in range(0,4):
69          for j in range(0,4):
70              column[i]+=confusion_matrix[j][i] #求出每列的和
71      for i in range(0,4):
72          for j in range(0,4):
73              line[i]+=confusion_matrix[i][j]  #求出每行的和
74      for i in range(0,4):
75          accuracy += float(accu[i])/totalNumOfTestData #计算accuracy
76      for i in range(0,4):
77          if column[i] != 0:
78              precision +=float(accu[i])/column[i]  #每种类别precision之和
79      precision = precision / 4  # precision的算数平均
80      for i in range(0,4):
81          if line[i] != 0:
82              recall +=float(accu[i])/line[i]   #每种类别recall之和
83      recall = recall / 4  # recall的算数平均
84      f1_score = (2 * (precision * recall)) / (precision + recall) #计算F1
85      print('the totalNumOfTestData is %s' % totalNumOfTestData)
86      print('the accu is %s' % accu)
87      print('accuracy  = %s' % accuracy)
88      print('precision = %s' % precision)
89      print('recall    = %s' % recall)
90      print('f1_score  = %s' % f1_score)
```

六、分类器测试

6.1 分类器的训练与测试代码

通过数据预处理，将数据顺序随机打乱，然后选取 80%数据作为训练数据集，20%作为测试数据集。然后将数据集的属性值与 label 分离，并且将属性值进行归一化。将数据传入分类器，并通过自己实现的 getConfusionMatrix 与 getEvaluation 获取混淆矩阵以及评价指标。

```
92  #分类器训练与测试
93  def carEvaClassTest():
94      basePer = 0.2 #测试基数，选取文本中20%的数据进行测试
95      CarDataMat, CarLabels = dataPreprocess.dataCarMatrix()
96      normMat, ranges, minVals = dataPreprocess.autoNorm(CarDataMat) #进行数据归一化
97      totalLength = normMat.shape[0]  #读取数据的列长度
98      numTestVecs = int(totalLength * basePer) #确定测试的数量
99      confusion_matrix=[
100     [0,0,0,0],
101     [0,0,0,0],
102     [0,0,0,0],
103     [0,0,0,0]]
104     for i in range(numTestVecs):  #进行循环测试
105         result = classifyCarWithKNN(normMat[i, :], normMat[numTestVecs:totalLength, :], \
106                                     CarLabels[numTestVecs:totalLength], 6) #通过分类器进行判断
107         confusion_matrix = getConfusionMatrix(confusion_matrix,CarLabels[i],result)
108     print('confusion_matrix  = %s' % confusion_matrix)
109     getEvaluation(confusion_matrix,numTestVecs)
```

6.2 分类器运行结果与评估指标

本次实验代码共计 3 个文件：

1) 数据集：car.data
2) 数据预处理代码：dataPreprocess.py
3) 分类器建立与测试代码：carEvalutionWithKNN.py

直接运行 python carEvalutionWithKNN.py 可以直接看到分类器评价指标的各个值。另外需要注意的是，实验的测试数据是每次随机从样本中选取 20%，所以每次运行本程序，获得的 accuracy、precision、recall 以及 F1 会有所不同。经过多次测试发现 accuracy 稳定在 0.96 左右。

```
Y@DESKTOP-FBS193I MINGW64 ~/Desktop/DataMining
$ python carEvalutionWithKNN.py
confusion_matrix  = [[240, 2, 0, 0],
                     [4, 71, 0, 0],
                     [0, 2, 14, 0],
                     [0, 4, 0, 8]]
the totalNumOfTestData is 345
the accu is [240, 71, 14, 8]
accuracy  = 0.9652173913043478
precision = 0.9705851836480598
```

```
recall    = 0.8700172176308539
f1_score  = 0.9175537534499361
```

七、实验总结

【数据处理时】

1) 在每个文件顶部标注变量/属性名称，而且最好清晰易懂，虽然在最后使用数据集的时候不会用到，但是在处理过程中会很有帮助；
2) 一个数据集单独保存在一个文件当中。尽量不要使用一个 Excel 文件放多个表；
3) 实验过程和方法分别记录下来，以便后期的分析；
4) 保留原始数据。处理了之后原始数据不要丢掉。

【分类器算法的实现】

1) 简单的算法可以尝试自己实现
2) 搞清楚各个算法的优缺点以及适用前提
3) 可以使用封装的库函数，但是要清楚原理

八、参考资料

[1] Brink, H., Richards, J., & Fetherolf, M. (2016). Real-world machine learning. Manning Publications Co..

[2] Awwalu, J., Ghazvini, A., & Bakar, A. A. Performance Comparison of Data Mining Algorithms: A Case Study on Car Evaluation Dataset.

[3] Daelemans, W., Hoste, V., De Meulder, F., & Naudts, B. (2003, September). Combined optimization of feature selection and algorithm parameters in machine learning of language. In European Conference on Machine Learning (pp. 84-95). Springer, Berlin, Heidelberg.

[4] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Vanderplas, J. (2011). Scikit-learn: Machine learning in Python. Journal of machine learning research, 12(Oct), 2825-2830.

[5] Wahbeh, A. H., Al-Radaideh, Q. A., Al-Kabi, M. N., & Al-Shawakfa, E. M. (2011). A comparison study between data mining tools over some classification methods. International Journal of Advanced Computer Science and Applications, 8(2), 18-26.

[6] UCI Machine Learning Group [online] ftp://ftp.ics.uci.edu/bar/machine-learning-databases.

[7] https://en.wikipedia.org/wiki/Confusion_matrix

[8] https://en.wikipedia.org/wiki/F1_score