实验一　贝叶斯分类器的设计及应用实验

实验目标：理解朴素贝叶斯分类器的原理；
能独立实现常用贝叶斯分类器的设计；
准确评估分类器精度。

实验工具：Python(推荐) 或 C/C++

实验步骤：

一、朴素贝叶斯分类算法原理理解

每个数据样本用一个 n 维特征向量 **X={x_1,x_2,…,x_n}**表示；分别描述对 n 个属性 A_1,A_2,..,A_n 样本的 n 个度量。假定有 **m 个类 C_1,…，C_m**,对于数据样本 X,分类法将预测 X 属于类 C_i,当且仅当：**$P(C_i|X) > P(C_j|X)$**,1<=j<=m 且 j≠i。

根据**贝叶斯定理**,：**$P(C_i|X)=P(X|C_i)P(C_i)/P(X)$**

只需最大化 $P(X \mid c_i)P(c_i)$ 。

假设属性之间相互独立，则 $P(X \mid c_j) = \prod_{i=1}^{n} P(x_i \mid c_j)$

一个对象被标记为 c_j，如果这个类是如下贝叶斯公式的分子取得最大值，则一个对象将标记为类。

$$P(c_j) \bullet \prod_{i=1}^{n} P(x_i \mid c_j)$$

三种常用模型：高斯模型、多项式模型、贝努利模型。

二、基于经典数据集实现糖尿病案例预测实验

1. 数据集简介

数据集 **pima-indians-diabetes.data(**详见附件，属于“皮马印第安人糖尿病问题”**)**，其中包括 768 个对于皮马印第安患者的医疗观测细节，记录所描述的瞬时测量取自诸如患者的年纪，怀孕和血液检查的次数。所有患者都是 21 岁以上（含 21 岁）的女性，所有属性都是数值型，而且属性的单位各不相同。

数据集的前 8 列分别记录怀孕次数、口服葡萄糖耐量试验中 2 小时血浆葡萄糖浓度、舒张压、三头肌皮褶厚度、2 小时血清胰岛素、体重指数 kg/m^2、糖尿病家族作用、年龄。

每一个记录归属于一个类，这个类指明以测量时间为止，患者是否是在 5 年之内感染的糖尿病。如果是，则为 1，否则为 0。

2. 朴素贝叶斯算法实验过程：

① 处理数据：从 CSV 文件中载入数据，然后划分为训练集和测试集。
② 提取数据特征：提取训练数据集的属性特征，以便我们计算概率并做出预测。
③ 单一预测：使用数据集的特征生成单个预测。
④ 多重预测：基于给定测试数据集和一个已提取特征的训练数据集生成预测。
⑤ 评估精度：评估对于测试数据集的预测精度作为预测正确率。
⑥ 合并代码：使用所有代码呈现一个完整的、独立的朴素贝叶斯算法的实现。

1) 处理数据

首先加载数据文件，使用 csv 模块中的 open 函数打开文件，使用 reader 函数读取行数据。

```
def loadCsv(filename):
    lines = csv.reader(open(filename, "rb"))
    dataset = list(lines)
    for i in range(len(dataset)):
        dataset[i] = [float(x) for x in dataset[i]]
    return dataset
```

可以通过加载皮马印第安人数据集，然后打印出数据样本的个数。

```
filename = 'pima-indians-diabetes.data.csv'
dataset = loadCsv(filename)
print('Loaded data file {0} with {1} rows').format(filename, len(dataset))
```

下一步，用 splitDataset()函数将数据分为用于朴素贝叶斯预测的训练数据集（67%）和测试数据集（33%）。

```
import random


def splitDataset(dataset, splitRatio):
    trainSize = int(len(dataset) * splitRatio)
    trainSet = []
    copy = list(dataset)
    while len(trainSet) < trainSize:
        index = random.randrange(len(copy))
        trainSet.append(copy.pop(index))
    return [trainSet, copy]
```

可以定义一个具有 5 个样例的数据集来进行测试，把它分为训练数据集和测试数据集，打印出来，看每个数据样本最终落在哪个数据集。

```
dataset = [[1], [2], [3], [4], [5]]
splitRatio = 0.67
train, test = splitDataset(dataset, splitRatio)
print('Split {0} rows into train with {1} and test with {2}').format(len(dataset), train, test)
```

2) 提取数据特征

朴素贝叶斯模型包含训练数据集中数据的特征，然后使用这个数据特征来做预测。

所收集的训练数据的特征，包含相对于每个类的每个属性的均值和标准差。例如，有 2 个类和 7 个数值属性，则需要每一个属性（7）和类（2）的组合的均值和标准差，即 14 个属性特征。

我们将数据特征的获取划分为以下的子任务：

a. 按类别划分数据
b. 计算均值
c. 计算标准差
d. 提取数据集特征
e. 按类别提取属性特征

划分数据

SeparateByClass()函数按类别划分数据，然后计算出每个类的统计数据。

```
def separateByClass(dataset):
    separated = {}
    for i in range(len(dataset)):
        vector = dataset[i]
        if (vector[-1] not in separated):
            separated[vector[-1]] = []
        separated[vector[-1]].append(vector)
    return separated
```

可以用一些样本数据测试如下：

```
dataset = [[1, 20, 1], [2, 21, 0], [3, 22, 1]]
separated = separateByClass(dataset)
print('Separated instances: {0}').format(separated)
```

计算均值

计算在每个类中每个属性的均值、每个类中每个属性的标准差。均值是数据的中点或者集中趋势，在计算概率时，用来作为高斯分布的中值。

```
def mean(numbers):
    return sum(numbers) / float(len(numbers))

def stdev(numbers):
    avg = mean(numbers)
    variance = sum([pow(x - avg, 2) for x in numbers]) / float(len(numbers) - 1)
    return math.sqrt(variance)
```

提取数据集的特征

对于一个给定的样本列表（对应于某个类），我们可以计算每个属性的均值和标准差。

```
def summarize(dataset):
    summaries = [(mean(attribute), stdev(attribute)) for attribute in zip(*dataset)]
    del summaries[-1]
    return summaries
```

测试 summarize（）函数如下：

```
dataset = [[1, 20, 0], [2, 21, 1], [3, 22, 0]]
summary = summarize(dataset)
print('Attribute summaries: {0}').format(summary)
```

按类别提取属性特征

首先将训练数据集按照类别进行划分，然后计算每个属性的摘要

```
def summarizeByClass(dataset):
    separated = separateByClass(dataset)
    summaries = {}
    for classValue, instances in separated.iteritems():
        summaries[classValue] = summarize(instances)
    return summaries
```

使用小的测试数据集来测试 summarizeByClass()函数

```
dataset = [[1, 20, 1], [2, 21, 0], [3, 22, 1], [4, 22, 0]]
summary = summarizeByClass(dataset)
print('Summary by class value: {0}').format(summary)
```

3）预测

我们现在可以使用从训练数据中得到的摘要来做预测。做预测涉及到对于给定的数据样本，计算其归属于每个类的概率，然后选择具有最大概率的类作为预测结果。

我们可以将这部分划分成以下任务：

a. 计算高斯概率密度函数
b. 计算对应类的概率
c. 单一预测
d. 多重预测
e. 评估精度

计算高斯概率密度函数

给定来自训练数据中已知属性的均值和标准差，可以使用高斯函数来评估一个给定的属性值的概率，如下 calculateProbability()函数实现。

```
import math

def calculateProbability(x, mean, stdev):
    exponent = math.exp(-(math.pow(x - mean, 2) / (2 * math.pow(stdev, 2))))
    return (1 / (math.sqrt(2 * math.pi) * stdev)) * exponent
```

使用一些简单的数据测试如下：

```
x = 71.5
mean = 73
stdev = 6.2
probability = calculateProbability(x, mean, stdev)
print('Probability of belonging to this class: {0}').format(probability)
```

计算所属类的概率

合并一个数据样本中所有属性的概率，得到整个数据样本属于某个类的概率。

```
def calculateClassProbabilities(summaries, inputVector):
    probabilities = {}
    for classValue, classSummaries in summaries.iteritems():
        probabilities[classValue] = 1
        for i in range(len(classSummaries)):
            mean, stdev = classSummaries[i]
            x = inputVector[i]
            probabilities[classValue] *= calculateProbability(x, mean, stdev)
    return probabilities
```

测试 calculateClassProbabilities()函数如下：

```
summaries = {0: [(1, 0.5)], 1: [(20, 5.0)]}
inputVector = [1.1, '?']
probabilities = calculateClassProbabilities(summaries, inputVector)
print('Probabilities for each class: {0}').format(probabilities)
```

单一预测

找到最大的概率值，并返回关联的类。下面的 predict()函数实现。

```
def predict(summaries, inputVector):
    probabilities = calculateClassProbabilities(summaries, inputVector)
    bestLabel, bestProb = None, -1
    for classValue, probability in probabilities.iteritems():
        if bestLabel is None or probability > bestProb:
            bestProb = probability
            bestLabel = classValue
    return bestLabel
```

测试 predict()函数：

```
summaries = {'A': [(1, 0.5)], 'B': [(20, 5.0)]}
inputVector = [1.1, '?']
result = predict(summaries, inputVector)
print('Prediction: {0}').format(result)
```

多重预测

通过对测试数据集中每个数据样本的预测，我们可以评估模型精度。getPredictions()函数实现这个功能，并返回每个测试样本的预测列表。

```
def getPredictions(summaries, testSet):
    predictions = []
    for i in range(len(testSet)):
        result = predict(summaries, testSet[i])
        predictions.append(result)
    return predictions
```

测试 getPredictions()函数：

```
summaries = {'A': [(1, 0.5)], 'B': [(20, 5.0)]}
testSet = [[1.1, '?'], [19.1, '?']]
predictions = getPredictions(summaries, testSet)
print('Predictions: {0}').format(predictions)
```

评估精度

将预测值和测试数据集中的类别值进行比较，可以得到精确率作为分类的精确度。getAccuracy()函数可以计算出这个精确率。

```
def getAccuracy(testSet, predictions):
    correct = 0
    for x in range(len(testSet)):
        if testSet[x][-1] == predictions[x]:
            correct += 1
    return (correct / float(len(testSet))) * 100.0
```

4）合并代码

最后，将代码连贯起来，即朴素贝叶斯的 Python 逐步实现的全部代码。

三、在第二步基础上，更换数据集，独立实现红酒预测案例

数据集 **wine.data** 是对在意大利同一地区生产的三种不同品种的酒，做大量分析所得出的数据。这些数据包括了三种酒中 13 种不同成分的数量。分别为：Alcohol，Malicacid，Ash，

红酒数据集图示

Alcalinity of ash，Magnesium，Total phenols，Flavanoids，Nonflavanoid phenols，Proanthocyanins，Color intensity， Hue， OD280/OD315 of diluted wines， Proline。

在 “wine.data”文件中，每行代表一种酒的样本，共有 178 个样本；一共有 14 列，其中，第一列为类标志属性，共有三类，分别记为“1”，“2”，“3”；后面的 13 列为每个样本的对应属性的样本值。其中第 1 类有 59 个样本，第 2 类有 71 个样本，第 3 类有 48 个样本。

实验要求：用以上 GaussianNB 分类器结合 win.data 数据集，实现测试样本抽取，并对测试样本分类，与标签比对后统计分类预测精度。

注意：

1. **Python 的版本不同，涉及到语法的修改**
2. **两个数据集结构别，需要修改相应语句**
3. **运行出结果后，对比 sklearn 中封装的贝叶斯分类器使用，比较差异**

实验报告要求：

1. 为便于实验统计，文件名为学号_MachineLearning.rar 格式。

2. 以*.rar 格式提交，包括 Python 代码、运行结果截图及分析、代码说明文档、关键问题解决。

3. 下次实验课之前提交到教辅平台。

参考资料：http://www.jb51.net/article/63164.htm

https://blog.csdn.net/qq_22562949/article/details/49755413