**华中科技大学计算机科学与技术学院**

**机器学习报告**



专 业： 计算机科学与技术

班 级： 计算机1807班

学 号： U201814681

姓 名： 徐泽楠

成 绩：

指导教师： 邹复好

**完成日期： 2020年 6月21日**

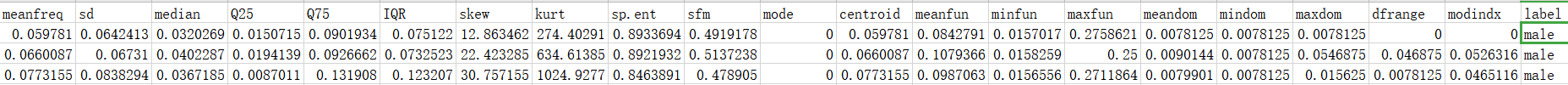
# 实验一

## 一、实验题目：基于朴素贝叶斯分类器的语音性别识别

## 二、实验要求

用朴素贝叶斯分类器进行数字手写体识别(基于MINIST数据集)，因此在这里用朴素贝叶斯在语音上做一个小应用——分辨声音是男性还是女性。使用7：3划分数据集。通过朴素贝叶斯方法，可以先对所有特征值做统计，并且通过连续性参数估计（高斯分布）方法得到参数。之后使用预测函数预测测试集。

## 三、算法设计



**图1-1 数据集预览**

S为性别，F为20个特征值的参数。

1. 划分数据集

将数据集进行随机排列后以7：3比例分出训练集和验证集

1. 计算先验概率

由数据集介绍可知，P(S=male)=P(S=female)=0.5。

对训练集进行统计可以得出P(A=male)和P(A=female)的统计数值，预测都约为0.5。

1. 计算类条件概率

计算P(F1|S=male)、P(F2|S=male)...P(F20|S=male)以及P(F1|S=female)、P(F2|S=female)...P(F20|S=female)这40种组合的高斯分布函数。首先分别计算每种属性的均值和方差，然后得出对应的高斯分布。

方差公式：

var=∑(x−avg)^2/(n−1)

概率密度函数：

p(xi|c)=[1/(√2π\*σc)]\*iexp(−(xi−meanc,i)^2/2σ^2) , σ是标准差(方差开方)

每个属性特征属于每类的条件概率是个组合,有两个类和20个数值属性，所以有2\*20种类属性条件概率以及对应的高斯分布。

得到类条件概率后按类别将每个属性的条件概率相乘。

1. 先验概率\*类条件概率

根据贝叶斯分类器的原理，后验概率在样本确定时，分母不变的情况下只和先验概率\*类条件概率这一分子有关，此时根据二者相乘即可实现贝叶斯分类的目的。

1. 分类准确率的计算

对测试集分男女两类别分别采取由训练集得出的贝叶斯分类器，将分类结果得到的性别与实际性别对比得到准确的数量N，在测试的过程计算总量M。

男性/女性分类准确率=N/M\*100%

男性/女性分类不准确率=1-N/M\*100%

## 四、实验环境与平台

环境：Miniconda

平台：jupyter notebook

## 五、程序实现

1.Python源代码

# coding = utf-8

import pandas as pd

import numpy as np

import random

import math

import matplotlib

from matplotlib import pyplot as plt

class bayesianClassifier(object):

def \_\_init\_\_(self, ratio=0.7):

self.trainset = []

self.testset = []

self.ratio = ratio

def loadData(self, filepath):

"""

:param filepath: csv

:return: list

"""

data\_df = pd.read\_csv(filepath)

data\_list = np.array(data\_df).tolist()

qlist=[]

for i in range(0,20):

a=[]

for j in range(0,len(data\_list)):

a.append(data\_list[j][i])

qlist.append(a)

for i in range(0,20):

for j in range(0,len(data\_list)):

data\_list[j][i]=math.floor(10\*(data\_list[j][i]-min(qlist[i]))/(max(qlist[i])-min(qlist[i])))

#量化数据

print("Loaded {0} samples sucessfully.".format(len(data\_list)))

self.trainset, self.testset = self.splitData(data\_list)

return data\_list

def splitData(self, data\_list):

"""

:param data\_list:all data with list type

:param ratio: train date's ratio

:return: list type of trainset and testset

"""

train\_size = int(len(data\_list) \* self.ratio)

random.shuffle(data\_list)

self.trainset = data\_list[:train\_size]

self.testset = data\_list[train\_size:]

return self.trainset, self.testset

def seprateByClass(self, dataset):

"""

:param dataset: train data with list type

:return: seprate\_dict:separated data by class;

info\_dict:Number of samples per class(category)

"""

seprate\_dict = {}

info\_dict = {}

for vector in dataset:

if vector[-1] not in seprate\_dict:

seprate\_dict[vector[-1]] = []

info\_dict[vector[-1]] = 0

seprate\_dict[vector[-1]].append(vector)

info\_dict[vector[-1]] += 1

return seprate\_dict, info\_dict

def mean(self, number\_list):

number\_list = [float(x) for x in number\_list] # str to number

return sum(number\_list) / float(len(number\_list))

#均值

def var(self, number\_list):

number\_list = [float(x) for x in number\_list]

avg = self.mean(number\_list)

var = sum([math.pow((x - avg), 2) for x in number\_list]) / float(len(number\_list) - 1)

return var

#方差

def summarizeAttribute(self, dataset):

"""

calculate mean and var of per attribution in one class

:param dataset: train data with list type

:return: len(attribution)'s tuple ,that's (mean,var) with per attribution

"""

dataset = np.delete(dataset, -1, axis=1) # delete label

summaries = [(self.mean(attr), self.var(attr)) for attr in zip(\*dataset)]

return summaries

#计算每个属性的均值和方差

def summarizeByClass(self, dataset):

"""

calculate all class with per attribution

:param dataset: train data with list type

:return: num:len(class)\*len(attribution)

{class1:[(mean1,var1),(),...],class2:[(),(),...]...}

"""

dataset\_separated, dataset\_info = self.seprateByClass(dataset)

summarize\_by\_class = {}

for classValue, vector in dataset\_separated.items():

summarize\_by\_class[classValue] = self.summarizeAttribute(vector)

return summarize\_by\_class

def calulateClassPriorProb(self, dataset, dataset\_info):

"""

calculate every class's prior probability

:param dataset: train data with list type

:param dataset\_info: Number of samples per class(category)

:return: dict type with every class's prior probability

"""

dataset\_prior\_prob = {}

sample\_sum = len(dataset)

for class\_value, sample\_nums in dataset\_info.items():

dataset\_prior\_prob[class\_value] = sample\_nums / float(sample\_sum)

return dataset\_prior\_prob

def calculateProb(self, x, mean, var):

"""

Continuous value using probability density function as class conditional probability

:param x: one sample's one attribution

:param mean: trainset's one attribution's mean

:param var: trainset's one attribution's var

:return: one sample's one attribution's class conditional probability

"""

exponent = math.exp(math.pow((x - mean), 2) / (-2 \* var))

p = (1 / math.sqrt(2 \* math.pi \* var)) \* exponent

return p

#由于特征种类太多，故采取概率取对数的方法来减小误差

def calculateClassProb(self, input\_data, train\_Summary\_by\_class):

"""

calculate class conditional probability through multiply

every attribution's class conditional probability per class

:param input\_data: one sample vectors

:param train\_Summary\_by\_class: every class with every attribution's (mean,var)

:return: dict type , class conditional probability per class of this input data belongs to which class

"""

prob = {}

p = 1

for class\_value, summary in train\_Summary\_by\_class.items():

prob[class\_value] = 1

for i in range(len(summary)):

mean, var = summary[i]

x = input\_data[i]

p = self.calculateProb(x, mean, var)

p

prob[class\_value] \*= p

return prob

def bayesianPredictOneSample(self, input\_data):

"""

:param input\_data: one sample without label

:return: predicted class

"""

train\_separated, train\_info = self.seprateByClass(self.trainset)

prior\_prob = self.calulateClassPriorProb(self.trainset, train\_info)

train\_Summary\_by\_class = self.summarizeByClass(self.trainset)

classprob\_dict = self.calculateClassProb(input\_data, train\_Summary\_by\_class)

result = {}

for class\_value, class\_prob in classprob\_dict.items():

p = class\_prob \* prior\_prob[class\_value]

result[class\_value] = p

return max(result, key=result.get)

def calculateFemaleAccByBeyesian(self, ratio=0.7):

"""

:param dataset: list type,test data

:return: acc

"""

self.ratio = ratio

correct = 0

FemaleNumber = 0

for vector in self.testset:

input\_data = vector[:-1]

label = vector[-1]

if label == 'female':

FemaleNumber += 1

result = self.bayesianPredictOneSample(input\_data)

if result == label:

correct += 1

return correct / FemaleNumber

def calculateMaleAccByBeyesian(self, ratio=0.7):

"""

:param dataset: list type,test data

:return: acc

"""

self.ratio = ratio

correct = 0

MaleNumber = 0

for vector in self.testset:

input\_data = vector[:-1]

label = vector[-1]

if label == 'male':

MaleNumber += 1

result = self.bayesianPredictOneSample(input\_data)

if result == label:

correct += 1

return correct / MaleNumber

if \_\_name\_\_ == "\_\_main\_\_":

%matplotlib inline

#通用设置

matplotlib.rc('axes', facecolor = 'white')

matplotlib.rc('figure', figsize = (20, 20))

matplotlib.rc('axes', grid = False)

#数据及线属性

x=list(range(1,20+1))

y\_1=[]

y\_2=[]

for i in range (0,20):

bys = bayesianClassifier()

data\_samples = bys.loadData('D:\Voice.csv')

print(i,"Female:",bys.calculateFemaleAccByBeyesian(ratio=0.7))

print(i,"Male",bys.calculateMaleAccByBeyesian(ratio=0.7))

y\_1.append(bys.calculateFemaleAccByBeyesian(ratio=0.7))

y\_2.append(bys.calculateMaleAccByBeyesian(ratio=0.7))

plt.plot(x,y\_1,label="Female",color="#F08080")

plt.plot(x,y\_2,label="Male",color="#DB7093",linestyle="--")

avg\_1=sum(y\_1)/len(y\_1)

avg\_2=sum(y\_2)/len(y\_2)

print("Average FemaleAccuracy is:", avg\_1)

print("Average MaleAccuracy is:", avg\_2)

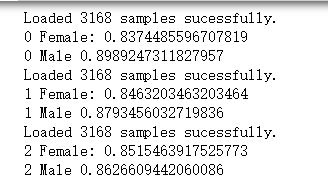
#标题设置

plt.title('Acc Table')

plt.xlabel('times')

plt.ylabel('Accuracy')

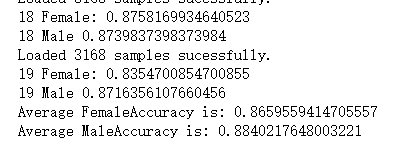
## 六、实验结果



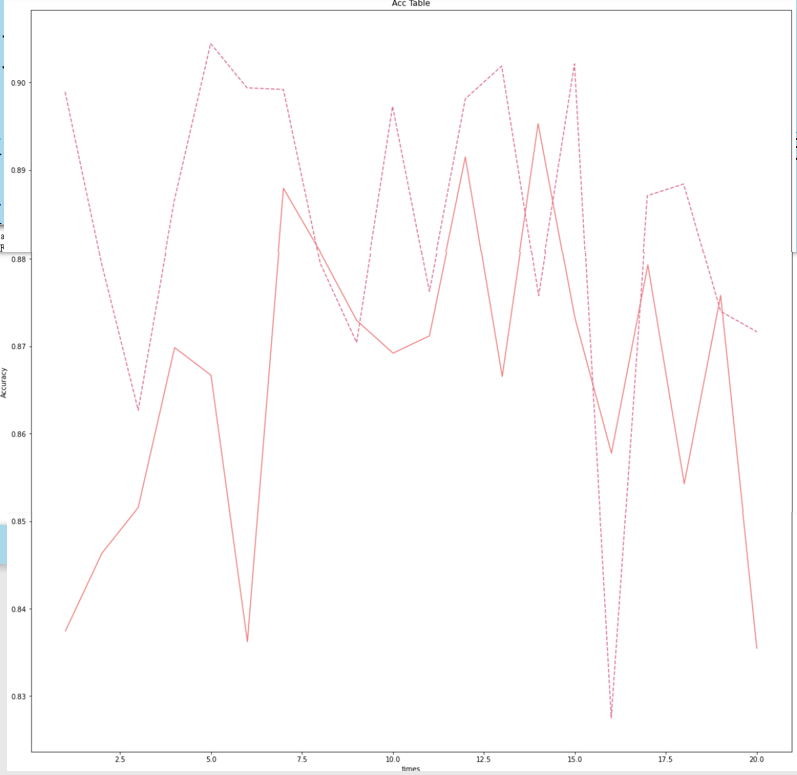
.............

.............

.............



**图1-2 实验结果截图1**



**图1-3 实验结果截图2**

## 七、结果分析

|  |  |
| --- | --- |
| 男声正确率:88.40% | 男声错误率:11.60% |
| 女声正确率:86.60% | 女声错误率:13.40% |

**表1-1 实验结果分析表**

# 参考文献

[1]数据集<https://www.kaggle.com/primaryobjects/voicegender/data>

[2]《机器学习内部讲义》

[3]《机器学习实战》

[4]<https://blog.csdn.net/leilei7407/article/details/103856584>