**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1804班

学 号： U201814596

姓 名： 左振

成 绩：

指导教师： 邹复好

**完成日期：2020年6月24日**

目录

[一、项目题目 3](#_Toc44509969)

[二、项目概述 3](#_Toc44509970)

[题目背景 3](#_Toc44509971)

[数据集 3](#_Toc44509972)

[任务描述 3](#_Toc44509973)

[评测标准 3](#_Toc44509974)

[三、算法设计 4](#_Toc44509975)

[高斯朴素贝叶斯算法原理简述 4](#_Toc44509976)

[具体实现过程 4](#_Toc44509977)

[拓展分析构想 5](#_Toc44509978)

[四、实验环境与平台 6](#_Toc44509979)

[五、程序实现 6](#_Toc44509980)

[高斯朴素贝叶斯python源代码 6](#_Toc44509981)

[拓展分析中基于sklearn的四种分类器实现的python源代码 10](#_Toc44509982)

[六、实验结果 13](#_Toc44509983)

[七、结果分析 14](#_Toc44509984)

[八、拓展分析 15](#_Toc44509985)

[抽样方式比较 15](#_Toc44509986)

[各个属性的分析 16](#_Toc44509987)

[特征筛选后的结果 19](#_Toc44509988)

[多种分类器实现及比较 20](#_Toc44509989)

[参考文献 22](#_Toc44509990)

## 一、项目题目

基于朴素贝叶斯分类器的语音性别识别

## 二、项目概述

### 题目背景

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。具体题目可以参考<https://www.kaggle.com/primaryobjects/voicegender>

### 数据集

数据集可自行在<https://www.kaggle.com/primaryobjects/voicegender>下载或[附件](voicegender.zip)。这个数据集是基于对男女语音段进行合理的声音预处理而得到的语音特征(并不包含原始语音段)。集合中共有3168条数据，男女各1584条，每条数据可视作一个长度为21的一维数组。其中前20个数值是这条语音的20个特征值，这些特征值包括了语音信号的长度、基频、标准差、频带中值点/一分位频率/三分位频率等；最后一个数值是性别标记。元数据集中直接以字符串,即male和female进行标注。使用7：3划分数据集。

### 任务描述

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。

### 评测标准

要求得到2\*2预测情况

|  |  |
| --- | --- |
| 男声正确率 | 男声错误率 |
| 女声正确率 | 女声错误率 |

## 三、算法设计

### 高斯朴素贝叶斯算法原理简述

（1）朴素贝叶斯简单来说就是在相关概率已知的情况下，选择最优的类别标记的过程。那么我们就需要求得后验概率，也就是各个标记在已知属性组合x的情况下的概率，最大的即为最优的类别标记；

（2）根据贝叶斯定理可知，求后验概率的问题就可以转化为如何基于训练数据D来估计类先验概率P(C)和条件概率P(x|C)的问题；

（3）在朴素贝叶斯算法下是假设各个属性相互独立的，所以条件概率P(x|C)由原本的联合概率变成了多个独立同分布属性的条件概率的乘积，而类先验概率P(C)在包含充足独立同分布样本的训练集中，可以根据大数定律用各类样本出现的频率来估计；

（4）所以朴素贝叶斯分类器的训练过程就是基于训练集D来估计类先验概率P(C)，并为每个属性估计条件概率P(Xi|C)的过程，最终解释公式如图1所示：

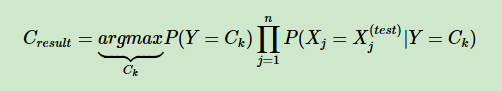


图1 朴素贝叶斯算法最终解释公式

（5）朴素贝叶斯在处理连续属性时考虑的是概率密度函数，就是假设这些属性均满足高斯分布，此时求解条件概率的公式如图2所示：

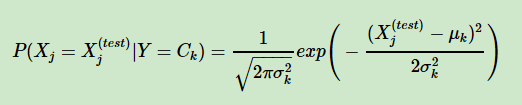


图2 属性满足高斯分布下求解条件概率的公式

### 具体实现过程

（1）处理数据集，按照题目的要求用训练集比测试集为7:3的比例来划分数据集，尝试实现分层抽样和随机抽样两种方式；

（2）求高斯分布的参数并构建字典，参数为求上述条件概率公式需要的各类样本在各个属性上取值的均值与方差；

（3）朴素贝叶斯过程，利用查字典的方式来找到目标参数；

（4）主模块以及交互过程，包括上述实现的汇总以及一些多次训练求平均值的功能。

### 拓展分析构想

（1）观察比较分层采样和随机采样两种采样方式的结果并分析；

（2）绘制构建字典之后的参数图，观察参数分布特点；

（3）绘制各个属性的分布图，观察属性分布特点；

（4）分别利用各个属性单独训练，看各个属性的表现并作比较；

（5）综合（2），（3）和（4）进行分析，进而实现特征筛选或是特征权重调整来提升正确率；

（6）用knn，svm，逻辑回归，高斯贝叶斯四种分类器来实现该问题，并做比较分析

## 四、实验环境与平台

（1）硬件环境：

处理器：Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz 1.99GHz

机带RAM：8.00GB

系统类型：64位操作系统，基于x64的处理器

（2）软件环境：

Windows 10 下 Anaconda Spyder

## 五、程序实现

### 高斯朴素贝叶斯python源代码

import numpy as np

import pandas as pd

import random

import collections

import math

from sklearn.metrics import confusion\_matrix

import matplotlib.pyplot as plt

import itertools

import time

#加载数据集

def data\_load(file\_name):

data\_set = pd.read\_csv(file\_name)

x = data\_set.values[:, :-1]

y = data\_set.values[:, -1]

'''

#随机划分训练集与测试集，比例7:3

train\_num = random.sample(range(0, 3167), 2218) #设置随机数生成从0-3167中随机挑选2218个随机数

test\_num = list(set(range(0, 3167)).difference(set(train\_num)))

train\_mat = np.array(x)[train\_num]

train\_label = np.array(y)[train\_num]

test\_mat = np.array(x)[test\_num]

test\_label = np.array(y)[test\_num]

'''

#分层采样

male\_train\_num = random.sample(range(0, 1583), 1109)

male\_test\_num = list(set(range(0, 1583)).difference(set(male\_train\_num)))

female\_train\_num = random.sample(range(1584, 3167), 1109)

female\_test\_num = list(set(range(1584, 3167)).difference(set(female\_train\_num)))

train\_mat = np.append(np.array(x)[male\_train\_num], np.array(x)[female\_train\_num], axis = 0)

train\_label = np.append(np.array(y)[male\_train\_num], np.array(y)[female\_train\_num], axis = 0)

test\_mat = np.append(np.array(x)[male\_test\_num], np.array(x)[female\_test\_num], axis = 0)

test\_label = np.append(np.array(y)[male\_test\_num], np.array(y)[female\_test\_num], axis = 0)

return train\_mat, train\_label, test\_mat, test\_label

#求高斯分布需要的参数并构建字典

def get\_para(train\_mat, train\_label):

male\_list = [] #男声的序号

female\_list = [] #女声的序号

L\_train = len(train\_label)

for i in range(L\_train):

if train\_label[i] == 'male':

male\_list.append(i)

else:

female\_list.append(i)

continuousPara = {}

for i in range(20):

fea\_data = train\_mat[male\_list, i]

mean = fea\_data.mean()

std = fea\_data.std()

continuousPara[(i, 'male')] = (mean, std)

fea\_data = train\_mat[female\_list, i]

mean = fea\_data.mean()

std = fea\_data.std()

continuousPara[(i, 'female')] = (mean, std)

return continuousPara

#绘制高斯分布需要的参数图

def show\_para(continuousPara):

feature\_mean = [] #记录各个特征在各标签下的均值

feature\_std = [] #记录各个特征在各标签下的方差

for key in continuousPara.keys():

feature\_mean.append(continuousPara[(key)][0])

feature\_std.append(continuousPara[(key)][1])

male\_mean = feature\_mean[::2]

female\_mean = feature\_mean[1::2]

male\_std = feature\_std[::2]

female\_std = feature\_std[1::2]

names = ['0','1','2','3','4','5','6','7','8','9'

,'10','11','12','13','14','15','16','17','18','19']

x = range(len(names))

plt.plot(x, male\_mean, marker = 'o', mec = 'r', mfc = 'w', label = 'mean under male')

plt.plot(x, female\_mean, marker = 'o', mec = 'r', mfc = 'w', label = 'std under male')

plt.plot(x, male\_std, marker = '\*', ms = 10, label = 'mean under female')

plt.plot(x, female\_std, marker = '\*', ms = 10, label = 'std under female')

plt.legend() #让图例生效

plt.xticks(x, names, rotation = 45)

plt.margins(0)

plt.subplots\_adjust(bottom = 0.15)

plt.xlabel("feature\_number") #X轴标签

plt.ylabel("Value") #Y轴标签

plt.title("Parameter") #标题

plt.show()

#正态分布函数

def gaussian(x, mean, std):

return 1 / (math.sqrt(math.pi \* 2) \* std) \* math.exp((-(x - mean) \*\* 2) / (2 \* std \* std))

#计算P(feature = x|C)

def P(feature\_Index, x, C, continuousPara):

fea\_para = continuousPara[(feature\_Index, C)]

mean = fea\_para[0]

std = fea\_para[1]

ans = gaussian(x, mean, std)

return ans

#高斯贝叶斯过程

def Bayes(X, train\_label, continuousPara):

#求先验概率

L\_train = len(train\_label)

num = collections.Counter(train\_label)

num = dict(num)

male\_para = num['male'] / L\_train

female\_para = num['female'] / L\_train

Result = []

L\_X = len(X)

for i in range(L\_X):

ans\_male = math.log(male\_para)

ans\_female = math.log(female\_para)

L\_Xi = len(X[i])

for j in range(L\_Xi):

#if j == 7|6|19:

#continue

ans\_male += math.log(P(j, X[i][j], 'male', continuousPara))

ans\_female += math.log(P(j, X[i][j], 'female', continuousPara))

'''

if j == 12:

ans\_male += math.log(P(j, X[i][j], 'male', continuousPara))

ans\_female += math.log(P(j, X[i][j], 'female', continuousPara))

'''

if ans\_male > ans\_female:

Result.append('male')

else:

Result.append('female')

return Result

#绘制混淆矩阵

def paintConfusion\_float(lmr\_matrix,classes):

plt.figure(figsize = (15, 10))

plt.imshow(lmr\_matrix, interpolation='nearest', cmap=plt.cm.Blues)

plt.title('confusion matrix')

plt.colorbar()

tick\_marks=np.arange(len(classes))

plt.xticks(tick\_marks, classes, rotation = 90, size = 18)

plt.yticks(tick\_marks, classes, size = 18)

plt.xlabel('Predict label', size = 20)

plt.ylabel('True label', size = 20)

lmr\_matrix = lmr\_matrix.astype('float') / lmr\_matrix.sum(axis = 1)[:,np.newaxis]

fmt='.6f'

thresh = lmr\_matrix.max() / 2.

for i, j in itertools.product(range(lmr\_matrix.shape[0]), range(lmr\_matrix.shape[1])):

plt.text(j, i, format(lmr\_matrix[i, j], fmt),

horizontalalignment = "center",

color = "red" if lmr\_matrix[i, j] > thresh else "black", size = 22)

plt.tight\_layout()

plt.show()

#主模块

def main\_():

train\_mat, train\_label, test\_mat, test\_label = data\_load('voice.csv') #加载数据集

continuousPara = get\_para(train\_mat, train\_label) #求高斯分布需要的参数并构建字典

show\_para(continuousPara) #绘制高斯分布需要的参数图

predict\_label = Bayes(test\_mat, train\_label, continuousPara) #高斯贝叶斯过程

confusionMatrix = confusion\_matrix(test\_label, predict\_label, labels = ['male', 'female']) #得出混淆矩阵

accuracy\_rate = (confusionMatrix[0][0] + confusionMatrix[1][1]) / len(test\_label)

classes=['male', 'female']

paintConfusion\_float(confusionMatrix, classes) #绘制混淆矩阵，显示男女声的正确率与错误率

return accuracy\_rate

if \_\_name\_\_ == '\_\_main\_\_':

accuracy\_rate = []

times = int(input("输入训练次数:\n"))

time\_start = time.time()

for i in range(times):

accuracy\_rate.append(main\_())

time\_end = time.time()

accuracy\_rate = np.array(accuracy\_rate)

print('accuracy\_rate: %f' %(accuracy\_rate.mean()))

print('Time used: %fs' %(time\_end - time\_start))

### 拓展分析中基于sklearn的四种分类器实现的python源代码

import pandas as pd

import time

import matplotlib.pyplot as plt

from sklearn.preprocessing import LabelEncoder

from sklearn.impute import SimpleImputer

from sklearn.model\_selection import train\_test\_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn import svm

from sklearn.linear\_model import LogisticRegression

from sklearn.naive\_bayes import GaussianNB

#数据准备

def data\_load(file\_name):

data\_set = pd.read\_csv(file\_name)

mat = data\_set.iloc[:,:-1]

label = data\_set.iloc[:,-1]

label = LabelEncoder().fit\_transform(label)

imp = SimpleImputer(missing\_values = 0, strategy = 'mean')

mat = imp.fit\_transform(mat)

train\_mat, test\_mat, train\_label, test\_label = train\_test\_split(mat, label, test\_size = 0.3)

return train\_mat, train\_label, test\_mat, test\_label

#KNN

def f1\_KNN(train\_mat, train\_label, test\_mat, test\_label):

best\_score = 0

for i in range(1,11):

knn\_clf = KNeighborsClassifier(n\_neighbors = i)

knn\_clf.fit(train\_mat, train\_label)

scores = knn\_clf.score(test\_mat, test\_label)

if scores > best\_score:

best\_score = scores

return best\_score

#SVM

def f2\_SVM(train\_mat, train\_label, test\_mat, test\_label):

cls = svm.LinearSVC()

cls.fit(train\_mat,train\_label)

return cls.score(test\_mat, test\_label)

#LogisticRegression

def f3\_LogisticRegression(train\_mat, train\_label, test\_mat, test\_label):

lr = LogisticRegression()

lr.fit(train\_mat, train\_label)

return lr.score(test\_mat, test\_label)

#GuassianNB

def f4\_GuassianNB(train\_mat, train\_label, test\_mat, test\_label):

gnb = GaussianNB()

gnb.fit(train\_mat, train\_label)

return gnb.score(test\_mat, test\_label)

if \_\_name\_\_ == '\_\_main\_\_':

times = [] #记录各算法时间消耗

scores = [] #记录各算法准确率

train\_mat, train\_label, test\_mat, test\_label = data\_load('voice.csv')

time\_start = time.time()

scores.append(f1\_KNN(train\_mat, train\_label, test\_mat, test\_label))

time\_end = time.time()

times.append(time\_end - time\_start)

time\_start = time.time()

scores.append(f2\_SVM(train\_mat, train\_label, test\_mat, test\_label))

time\_end = time.time()

times.append(time\_end - time\_start)

time\_start = time.time()

scores.append(f3\_LogisticRegression(train\_mat, train\_label, test\_mat, test\_label))

time\_end = time.time()

times.append(time\_end - time\_start)

time\_start = time.time()

scores.append(f4\_GuassianNB(train\_mat, train\_label, test\_mat, test\_label))

time\_end = time.time()

times.append(time\_end - time\_start)

#分别绘制时间消耗和准确率的柱状图

name\_list = ('KNN', 'SVM', 'LogisticRegression', 'GuassianNB')

plt.bar(name\_list, times, color = 'rbg')

plt.title('time used')

plt.xlabel('Arithmetic Name')

plt.ylabel('time/s')

for a, b in zip(name\_list, times):

plt.text(a, b, b, ha = 'center', va = 'bottom')

plt.show()

plt.bar(name\_list, scores, color = 'rbg')

plt.title('accuracy\_rate')

plt.xlabel('Arithmetic Name')

plt.ylabel('num')

for a, b in zip(name\_list, scores):

plt.text(a, b, '%.6f' % b, ha = 'center', va = 'bottom')

plt.show()

## 六、实验结果

（1）利用高斯分布构建的字典如下图3所示：

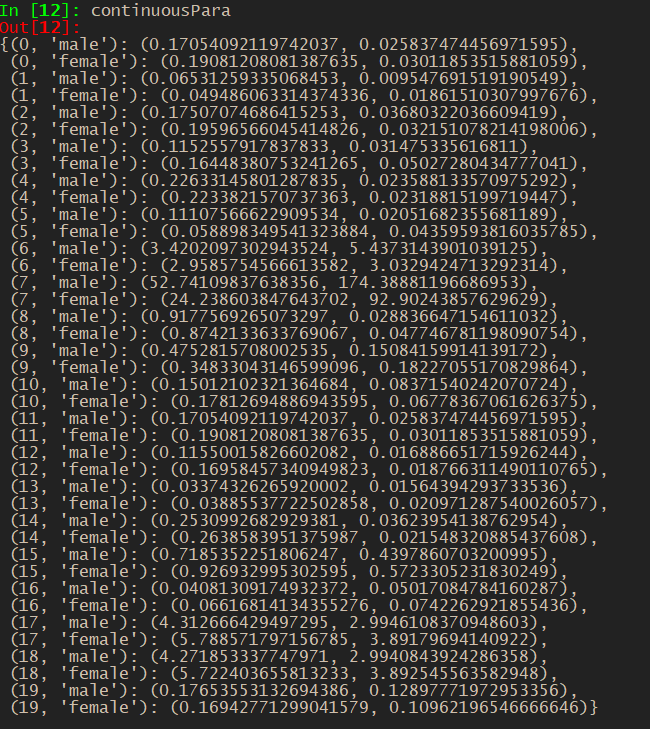


图3 参数字典

上图中0到19是属性标号，‘male’和‘female’是标签，后面的两列分别为各类样本在各个属性上取值的均值与方差。

（2）训练一次的正确率以及混淆矩阵如图4，5所示：

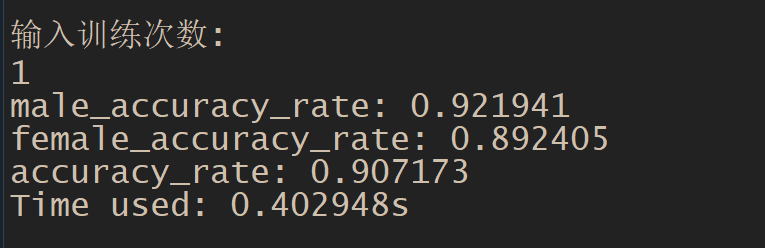


图4 训练一次正确率

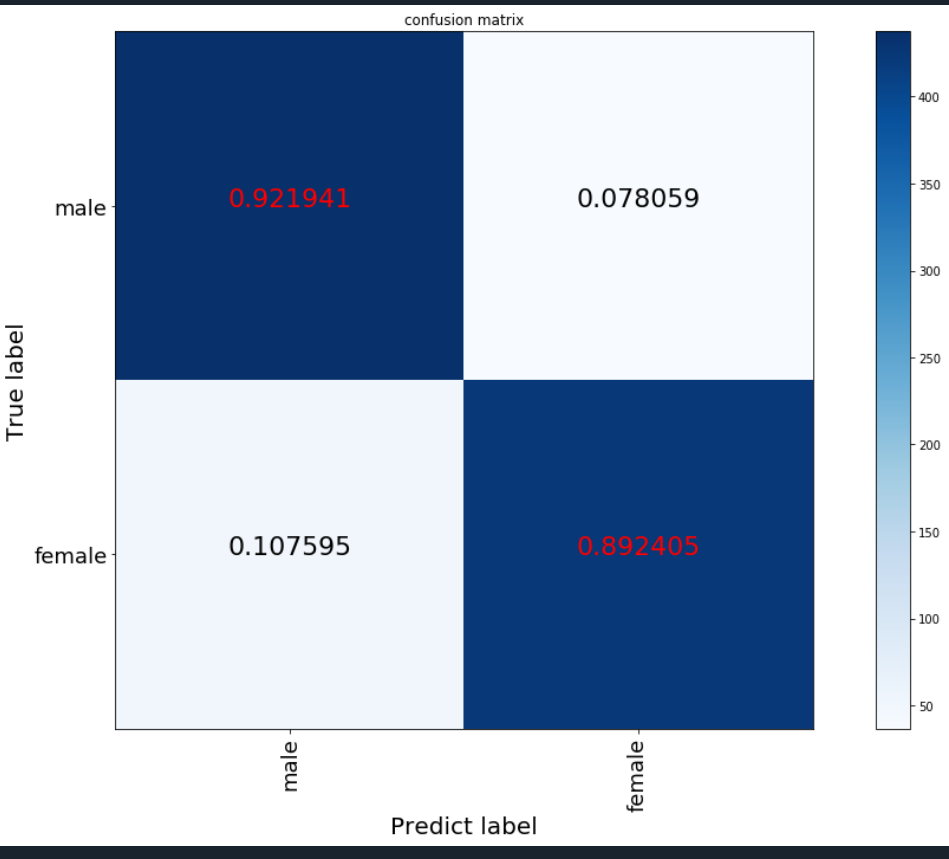


图5 混淆矩阵

（3）训练一百次的平均正确率如图6所示：

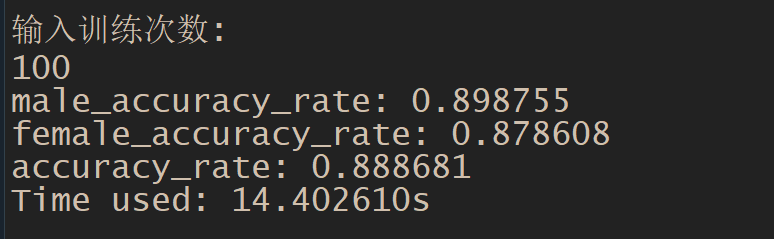


图6 训练一百次平均正确率

（4）由（3）得出实验结果如表1所示：

表1 实验结果

|  |  |  |
| --- | --- | --- |
|  | 正确率 | 错误率 |
| 男声 | 0.898755 | 0.101245 |
| 女声 | 0.878608 | 0.121392 |

## 七、结果分析

由上述实验结果可以得出总的正确率在89%左右，朴素贝叶斯算法的特点就是在刚实现完之后就容易到达一个正确率的上限，虽然在很多问题上朴素贝叶斯算法的表现很好，但是调参的余地不大，也就是提升空间不高。

接下来的拓展分析里会通过分析属性特点，尝试做一些改进（特征筛选）来提升正确率，并且用其他算法实现一下该分类问题，然后再做分类器比较。

## 八、拓展分析

### 抽样方式比较

这里尝试了两种抽样方式，第一种是直接在整个数据集里随机抽取7:3的训练集与测试集，第二种是在‘male’和‘female’两种标签下分别随机抽取7:3的训练集与测试集；分别训练100次观察结果，如图7，8所示：

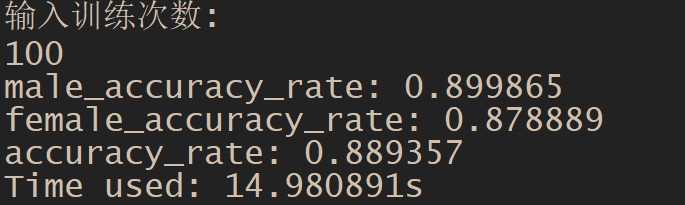


图7 随机抽样测试结果

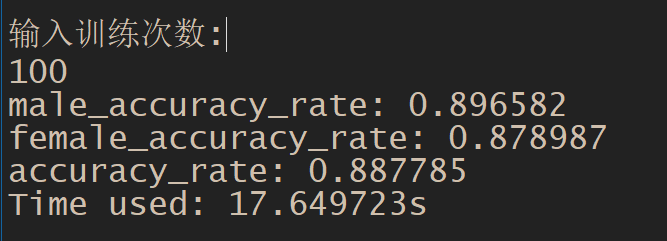


图8 分层抽样测试结果

由上图可知两种抽样方式在该分类问题下的表现基本上没什么区别，我认为这是朴素贝叶斯分类器本身特性的原因。因为朴素贝叶斯是基于严谨的数学公式来计算概率的，而且两种抽样方式所产生的影响就是先验概率的不同，也就是各个标签的出现频率不同，并且该数据集数据是比较充足的，极端的情况出现的概率极低，因此在这种情况下基本对朴素贝叶斯分类器没什么影响。

但是在其他的分类器中，分层抽样的表现会好一些，因为分层抽样所划分的数据分布较为均衡，能让分类器对各类标签做充分的学习。

### 各个属性的分析

（1）在利用高斯分布求得参数后绘制可视化图，如图9所示：

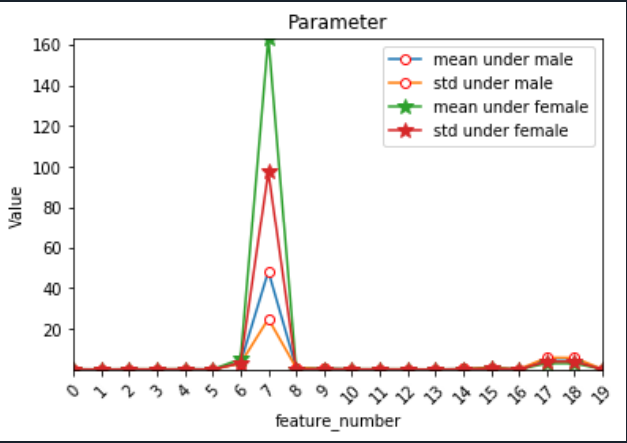


图9 参数折线图

由上图可知其中属性7在两个标签下的方差很大，说明该属性不具有普遍性，也就是说在人群中分布的差异很大，不适合作为分类判断依据；

（2）各个属性分别单独训练100次的平均正确率，如图10所示：

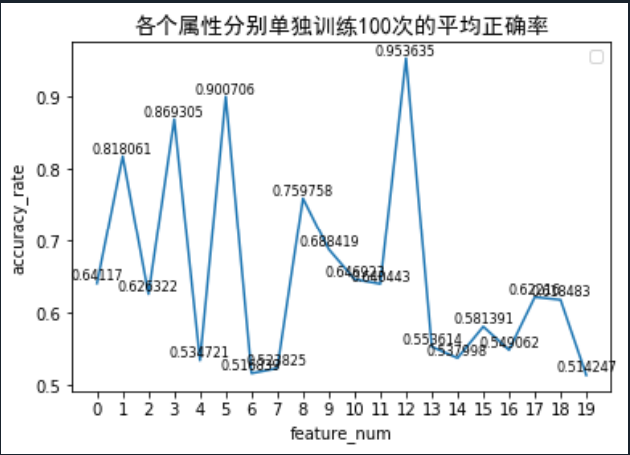


图10 各个属性单独训练的平均正确率

由图可知属性6，7，19等的表现是非常差的，而属性5，12的表现是很好的，超过了之前总的89%左右的正确率，甚至属性12单独训练的正确率达到了95%，接下来看一下这些有代表性的属性的数值分布图，观察一下实际的情况，如图11，12，13，14，15所示：

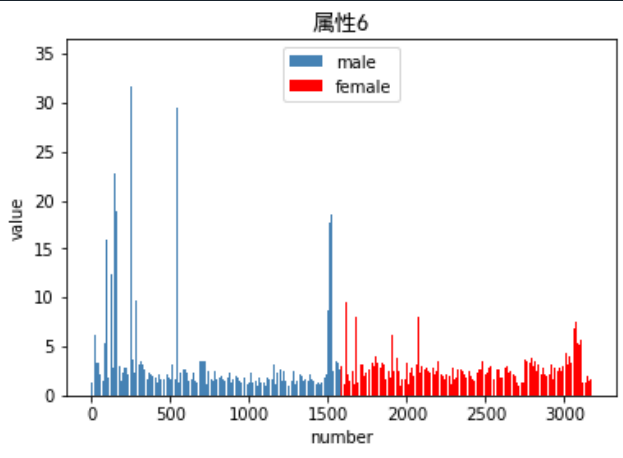


图11 属性6数值分布图

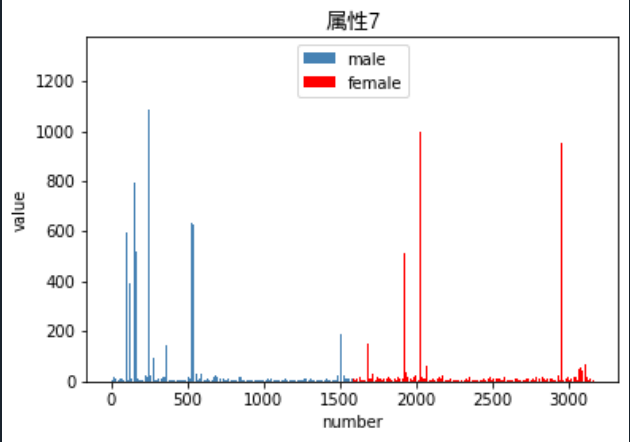


图12 属性7数值分布图

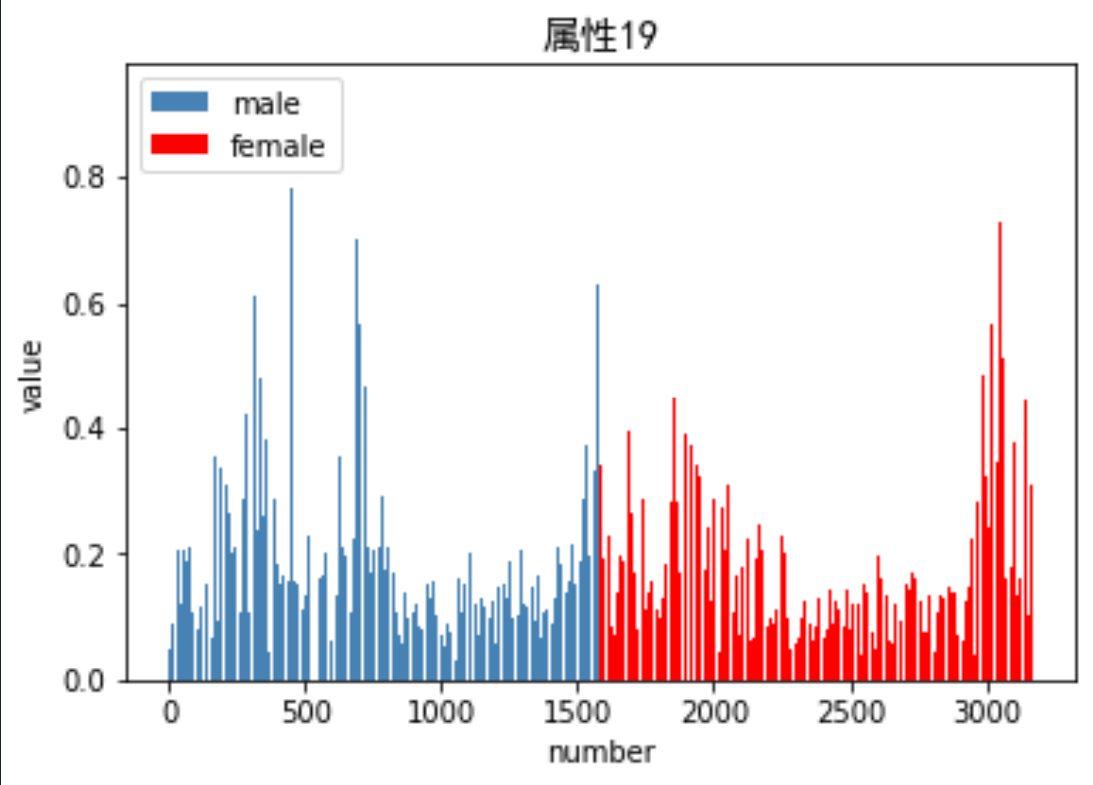


图13 属性19数值分布图

由上图11，12，13可以看出属性6，7和19在分类时表现不好的原因就是在男声，女声中的数据分布没有区分度，甚至属性7在整个人群里的分布都极不规律，这也是前面在计算参数时属性7对应的方差很大的原因。

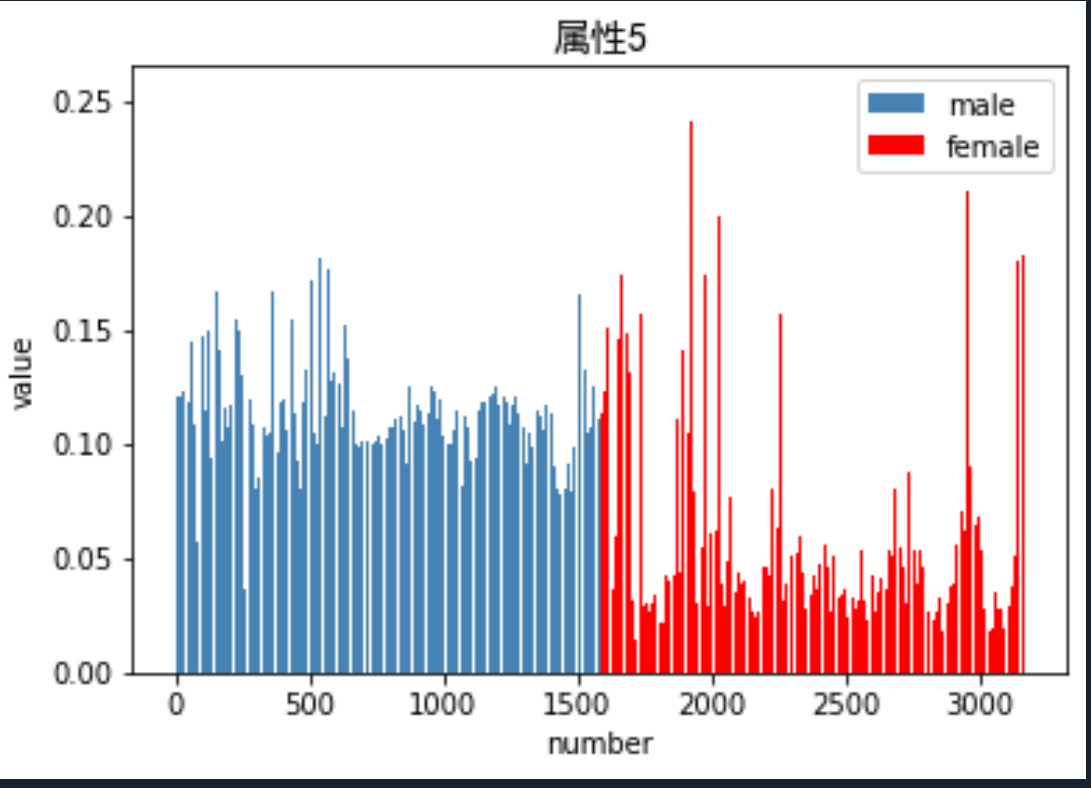


图14 属性5数值分布图

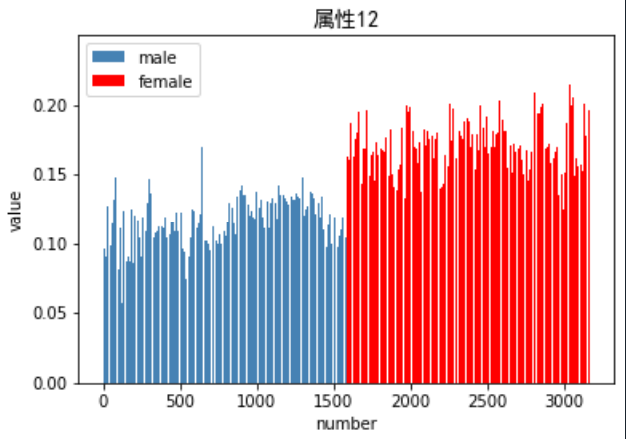


图15 属性12数值分布图

由上图14，15可知，属性5和属性12在分类时表现好的原因就在于在男声和女声里的分布很有区分度，特别是属性12体现得很明显。

（3）在做完属性的分析之后，可以尝试进行特征筛选来提升分类正确率，首先可以尝试删除掉属性6，7，19，看一下正确率改变；接下来可以尝试提升属性12的权重，再看一下正确率变化情况。

### 特征筛选后的结果

（1）删除掉属性6，7，19之后训练一百次如图16所示：

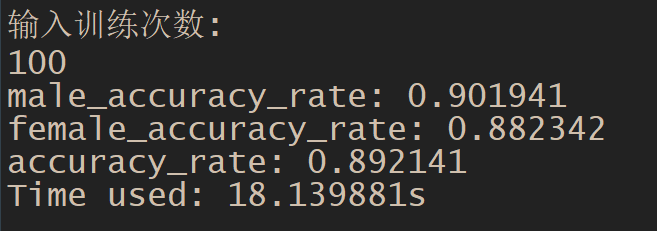


图16 删除属性6，7，19之后的结果

由上图可知正确率基本没有提升，由此可知这些干扰项对于分类器的影响不大。

（2）提高属性12的权重（具体实现就是引入两个新的属性，其实就是属性12），训练一百次如图17所示：

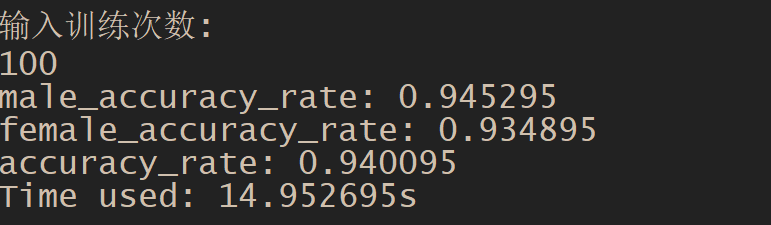


图17 提高属性12的权重之后的结果

由上图可以看见正确率提升的十分明显，如果继续提升属性12的权重的话显然会继续提升正确率直到95%左右，但是为了避免过拟合的情况就适可而止了。

（3）特征筛选之后的实验结果如表2所示：

表2 特征筛选后的实验结果

|  |  |  |
| --- | --- | --- |
|  | 正确率 | 错误率 |
| 男声 | 0.945295 | 0.054705 |
| 女声 | 0.934895 | 0.065105 |

在进行特征筛选之后总的正确率能提升到94%左右，实现的还是比较理想的，接下来尝试多种分类器实现，并作比较分析。

### 多种分类器实现及比较

（1）这里多种分类器都是基于sklearn来实现的，一共是四种适用于二分类的机器学习算法，即knn，svm，Logistic Regression，Naive Bayes；实现的python代码在“五.程序实现”部分；

（2）分类器的时间开销如图18所示：

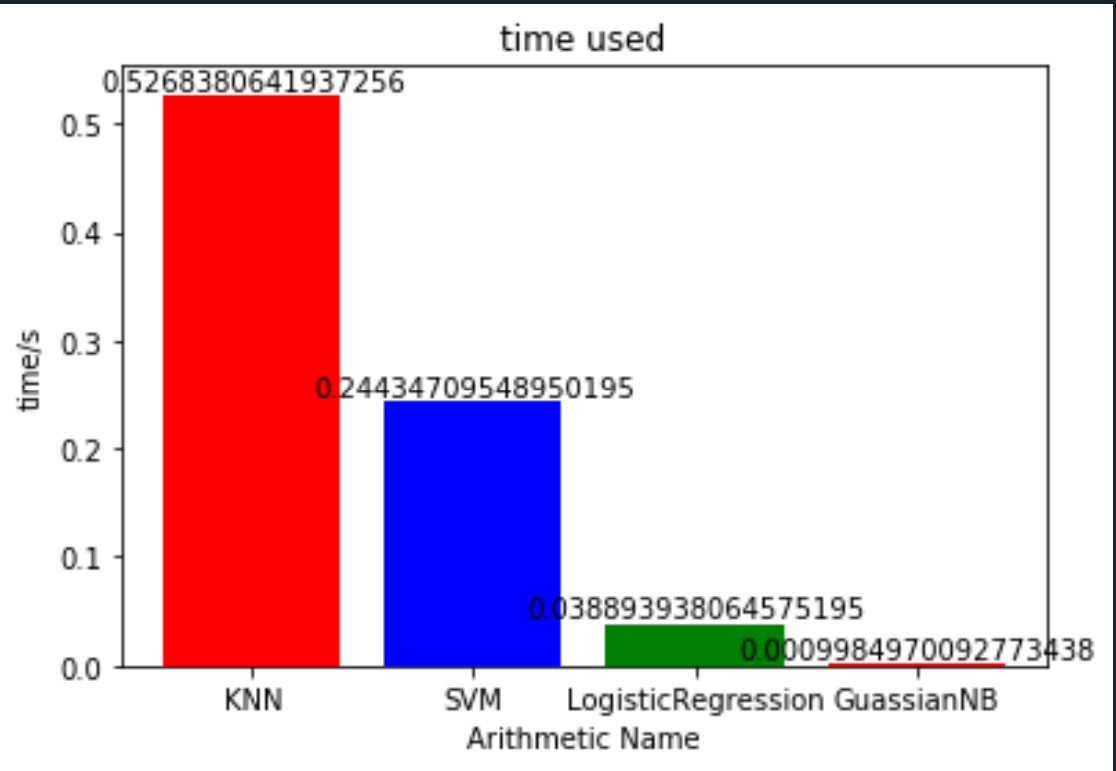


图18 分类器的时间开销

（3）分类器的准确率如图19所示：

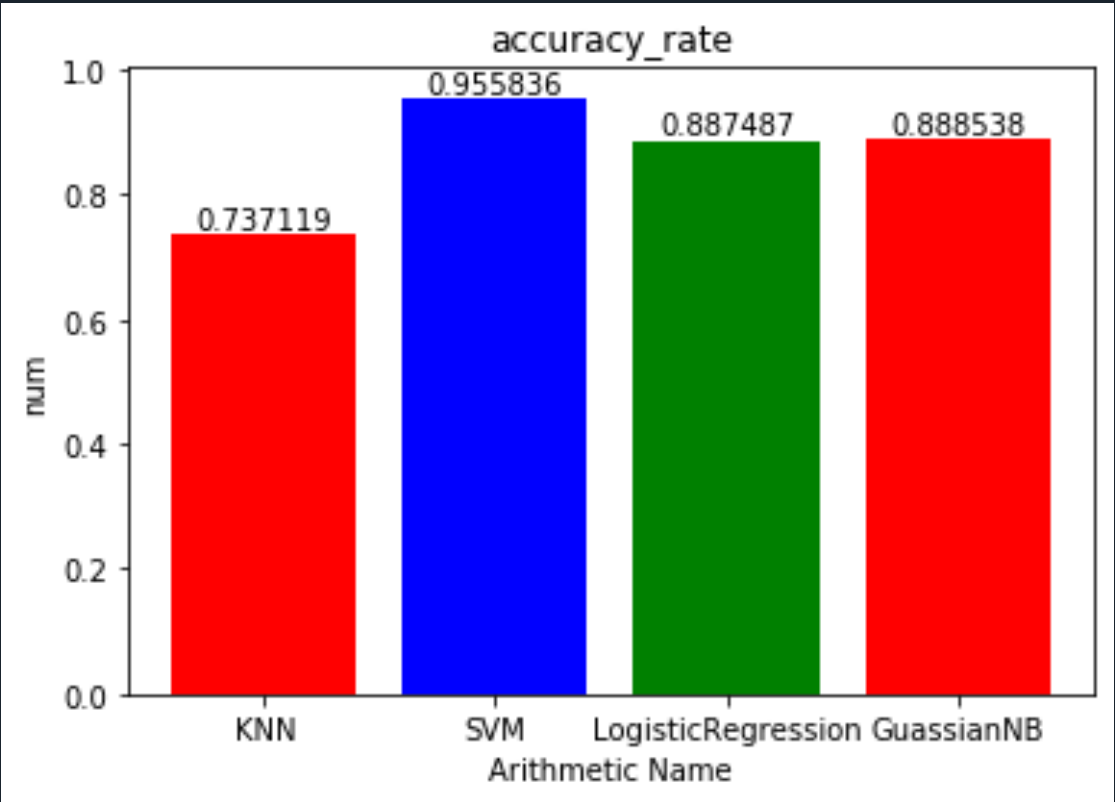


图19 分类器的准确率

（4）由图18可知，KNN分类器的时间开销最大，高斯贝叶斯的时间开销最小；由图19可知，SVM分类器的准确率最高，KNN分类器的准确率最低，逻辑回归和高斯朴素贝叶斯的准确率几乎相等。

KNN：由于KNN算法在每一次分类时都会进行一次全局运算，所以时间开销相对要大一些。在这个数据集上由于两类标签的样本容量是一样的，所以在这个方面不会对KNN的分类造成较大影响；KNN算法在二分类上的优势体现的不明显，在多分类问题下可能会较其余分类器有一定优势，主要的优势还是体现在算法容易理解并且对异常值不敏感上。

SVM：SVM分类器在这个数据集上的表现明显是优于其余三个分类器的，正确率能达到95.5%。主要在于经典的SVM算法就是处理二分类问题的，并且SVM的最终决策函数只由少数的支持向量所确定，这样就可以利用关键的样本来进行分类，进而有很好的二分类效果；

逻辑回归：LR在这个数据集上的表现还是很好的，时间开销不大并且最后的准确率也还不错。LR是一个很流行的算法，优势在于对资源的极少占用并且算法简单，但其本身无法筛选特征，对于数据不平衡的问题表现得不好；

高斯贝叶斯：GNB与LR在这个数据集上的表现差不多，时间开销少并且准确率可观。高斯贝叶斯在之前我已经分析过了，从理论上讲有概率论的支撑，分类效果是很不错的，但是对于独立性的假设往往是与事实不符的，并且满足于高斯分布也是一种经验假设。在某些实际问题中为了弱化这些假设的影响，会用到半朴素贝叶斯算法，也就是在求解条件概率时加上一个依赖属性，用这样的方式来实现改进。

（5）不同分类器往往是适用于不同的场景的，在这个数据集上的表现并不能真实的反应每个分类器的特点，具体问题还是要具体分析。

## 参考文献

[1] 周志华《机器学习》清华大学出版社

[2] <https://www.csdn.net/> CSDN-专业开发者社区

[3] [Magnus Lie Hetland](https://book.jd.com/writer/Magnus%20Lie%20Hetland_1.html) 著，[袁国忠](https://book.jd.com/writer/袁国忠_1.html) 译 《Python基础教程》（第三版）