模式识别实验（二）实验报告

学生信息

- 姓名-学号-班级
- 李欢欢-2020904302-2020240402
- 刘易行-2020905896-2020240401

实验题目

实验介绍：

朴素贝叶斯法是基于贝叶斯定理与特征条件独立假设的分类方法。请阅读教材 40 页**离散特征下的贝叶斯决策论**，尤其是 2.9.1 节**独立的二值特征**，理解对于连续数据和离散数据不同的概率表达方式及导致的具体分类算法的细节差异。

为了便于同学们入门，本次实验将采用基于独立二值特征的朴素（或简单）贝叶斯分类方法，实现手写体识别。采用的是 **MNIST** 数据库 MNIST Dataset | Kaggle，它是一个手写数学图片库，是美国普查员和高中学生的手写数字。其包含 60000 张训练图片和 10000 张测试图片，是常用的分类问题数据库。

具体算法实现可以参考 Python 手写实现朴素贝叶斯（从原理到代码） - 知乎 (zhihu.com)。（**提示：该参考内容中的伯努利算法就是我们所说的基于独立的二值特征的贝叶斯分类。如果下载的 MNIST 数据库图片的像素值是灰度值的，需要首先二值化为 0,1 离散数据，0 表示黑色，1 表示白色。**）

问题回答

（1）算法体现

3.5.1 步骤

1) 计算样本属于每个类的概率，在第二节中已经说明，只需求式中分子的值即可：

$$P(c_i|X=\{x_1,x_2,\ldots,x_n\}) = log(P(c_i)) + \sum_{j=1}^{n} log(P(x_j|c_i))$$

为什么要取对数？对每个 $P(x_j|c_i)$ 的计算结果有的非常接近0，由于计算机精度限制，多个接近0的数相乘，不利于大小的对比。

```
def ClassifySamples(sample,PCi,VEC_P):
    """
    Parameters:
    sample: 训练样本 10*784     类型:二维ndarray
    PCJ: 先验概率    10     类型:float64
    VEC_P: 即E矩阵
    """
    step1 = np.subtract(sample,1)  #数组数据相减
    step2 = np.abs(step1)  #求出绝对值
    step3 = np.subtract(step2,VEC_P)  #再相减
    step4=np.abs(step3) #求出最后的值
    log_P_Ci = np.log(PCi)
```

```
    log_P_Xj_Ci = np.log(step4)
    #计算P(Ci|x)=log(P(Ci))+Σlog(P(Xj|Ci))
    P_Ci_X = np.add(log_P_Ci,np.sum(log_P_Xj_Ci,axis=1))
    return np.argmax(P_Ci_X)
```

(2) (a)分类器是如何设计的?

- 此次实验基于书本P41页的独立二值特征的贝叶斯分类问题

数据处理

- 先来看下数据集的格式mnist_train.csv mnist_test.csv
- csv格式的数据集文件 本组采用python中的pandas库中的pd.readcsv()方法读取
- ```
  def getOriData():
      #读取原始CSV数据  未经处理
      train = pd.read_csv('mnist_train.csv',header=None)  #没有表头 header指定为None
      test = pd.read_csv('mnist_test.csv',header = None)
      return train,test
  ```
- 读取后，查看概览信息如下:
- ```
  trainCSV.info()
  ```

  ```
  <class 'pandas.core.frame.DataFrame'>
  RangeIndex: 60000 entries, 0 to 59999
  Columns: 785 entries, 0 to 784
  dtypes: int64(785)
  memory usage: 359.3 MB
  ```
- ```
  trainCSV
  ```

	0	1	2	3	4	5	6	7	8	9	...	775	776	777	778	779	780	781	782	783	7
0	5	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	
2	4	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	
3	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	
4	9	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
59995	8	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	
59996	3	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	
59997	5	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	
59998	6	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	
59999	8	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	

60000 rows × 785 columns

- 训练集共有60000行数据，每一条数据行的第一列为标签(label)，即为对应的手写数字
- 其余784行为特征，因为是28*28=784的位图格式，共计784个特征
  ```

- 显而易见，需要对数据进行切片处理，将数据条中的特征列和标签列分开，分成label和feature

```
train.loc[i][1:]=train.loc[i][1:].apply(Binarization)
#二维数组的切片不易理解 转换为一维切片更易理解
test.loc[i][1:]=test.loc[i][1:].apply(Binarization)
#拆分标签和特征向量
label = trainData.iloc[:,0]
feature = trainData.iloc[:,1:]
return label,feature
```

二值化映射

- 题目中指出，将数据特征进行二值化处理以方便分类，编写二值化映射函数如下：

```
#二值化映射函数
def Binarization(val):
    tag = 255/2
    if val < tag:
        return 0
    else:
        return 1
```

- 数据的二值化映射一开始编写的是双重for循环，时空复杂度过高，查阅资料，发现可以使用pandas库中的apply函数，对dataframe中的每一个数据执行函数映射
- 完整的数据处理函数如下：

```
def DataProcess(train,test):
    for i in range(train.shape[0]):
        #二维数组的切片不易理解 转换为一维切片更易理解
        train.loc[i][1:]=train.loc[i][1:].apply(Binarization)
    for i in range(test.shape[0]):
        #二维数组的切片不易理解 转换为一维切片更易理解
        test.loc[i][1:]=test.loc[i][1:].apply(Binarization)
    #拆分标签和特征向量
    label = trainData.iloc[:,0]
    feature = trainData.iloc[:,1:]
    return label,feature
```

独立二值特征朴素贝叶斯分类器

- 通过查阅互联网资料及书本P41页的独立二值特征朴素贝叶斯，原理概括如下：
- 独立二值特征：特征只有两种，此次实验中，图像的特征有且只有0，1两类 0表示黑色 1表示白色

$$P(x_j = 1|c_i) = \frac{I(x_j=1,c_i)}{I(c_i)}$$

- $$P(x_j = 0|c_i) = \frac{I(x_j=0,c_i)}{I(c_i)}$$

$I(x_j = 1, c_i)$ 表示第 j 个特征为1且类别为 c_i 的样本数。

- 在实际过程中，分母有可能出现计数为0的情况，这时，会抛出除0异常
- 因此引入拉普拉斯平滑

- $P(x_j=1|c_i)=\frac{I(x_j=1,c_i)+k}{I(c_i)+k*N}$

k 通常取值1-5之间， N 为特征所能取到的值的总数（伯努利特征只有0、1，所以设置为2）。

分类器公式

3.5.1 步骤

1) 计算样本属于每个类的概率，在第二节中已经说明，只需求式中分子的值即可：
$P(c_i|X=\{x_1,x_2,\ldots,x_n\})=log(P(c_i))+\sum_{j=1}^{n}log(P(x_j|c_i))$

为什么要取对数？对每个 $P(x_j|c_i)$ 的计算结果有的非常接近0，由于计算机精度限制，多个接近0的数相乘，不利于大小的对比。

计算分类器参数

- 计算先验概率（函数参数已给出完整注释）
- ```
  def getPCi(data):
      #计算先验概率 P(C|J)
      tag=0
      counts = [0]*10
      for i in range(data.shape[0]):
          tag=data.loc[i][0]
          counts[tag]+=1
      PCi=np.zeros(10)  #先验概率
      for i in range(10):
          PCi[i] = counts[i]/60000
      return counts,PCi
  ```
- 分类器训练函数（函数参数已给出完整注释）
- ```
  def TrainModel(df_feature,df_label,counts,k,n,labelNum):
      """
      Parameters:
      df_feature: 特征向量      类型:DataFrame  二值特征 0，1
      df_label: 标签向量        类型:DataFrame 其中值取0-9
      k:拉普拉斯平滑参数      类型:int
      counts:各个类别的样本数量 类型:array
      N:特征种数  本题为2
      labelNum:类别种数         类型:int
      """
      vecc = []   #存放的是每一个的P(Xj|Ci) 10*784
      for i in range(labelNum):
          print("当前正在训练的类别为:手写数字{}类".format(i))
          print("计算I(xj=1,c{})".format(i))
          I = feature[df_label==i].apply(np.sum,axis=0)  #获得每一个特征列中样本特征取值为1的个数
          p=(I+k) / (counts[i]+k*n)  #运用拉普拉斯平滑 计算P(Xj|Ci)
          vecc.append(p)
          print("I(xj=1,c{})={}".format(i,vecc[i][1]))
          print("手写数字{}类训练完成".format(i))
      return vecc
  ```
- 分类函数（函数参数已给出完整注释）

- ```
  def ClassifySamples(sample,PCi,VEC_P):
      """
      Parameters:
      sample: 训练样本 10*784     类型:二维ndarray
      PCJ: 先验概率    10     类型:float64
      VEC_P: 即E矩阵
      """
      step1 = np.subtract(sample,1)  #数组数据相减
      step2 = np.abs(step1)  #求出绝对值
      step3 = np.subtract(step2,VEC_P)  #再相减
      step4=np.abs(step3) #求出最后的值
      log_P_Ci = np.log(PCi)
      log_P_Xj_Ci = np.log(step4)
      #计算P(Ci|x)=log(P(Ci))+Σlog(P(Xj|Ci))
      P_Ci_X = np.add(log_P_Ci,np.sum(log_P_Xj_Ci,axis=1))
      return np.argmax(P_Ci_X)
  ```

(3) 对10000张测试图片进行测试，并计算算法识别准确度

- 分类测试函数如下：
- ```
  def ClassifyTestData(TestData,PCi,VEC_P):
      temp = []
      correctNum = 0
      FalseNum = 0
      for i in range(TestData.shape[0]):
          sample = TestData.iloc[i,1:].values
          MyAns = ClassifySamples(sample,PCi,VEC_P)
          Ans = TestData.loc[i][0]
          if i%5 == 0:
              print("Ans={},MyAns={}".format(Ans,MyAns))
          if MyAns == Ans:
              correctNum += 1
          else:
              FalseNum += 1
      print("CorrectNum={}".format(correctNum))
      print("FalseNum={}".format(FalseNum))
      correctRate = correctNum / (correctNum+FalseNum)
      return correctRate
  ```

```
vec = TrainModel(feature,label,counts,3,2,10)
```

- 当拉普拉斯平滑参数k取3时：准确度为0.8419
- ```
  Ans=2,MyAns=2
  CorrectNum=8419
  FalseNum=1581
  ```

  ```
  : Rate1
  ```

  ```
  : 0.8419
  ```
  ```

```
print("k=0, rate={}".format(r0))
print("k=1, rate={}".format(r1))
print("k=2, rate={}".format(r2))
print("k=3, rate={}".format(r3))
print("k=4, rate={}".format(r4))
print("k=5, rate={}".format(r5))
```

```
k=0, rate=0.8434
k=1, rate=0.8427
k=2, rate=0.8425
k=3, rate=0.8419
k=4, rate=0.8417
k=5, rate=0.8412
```

(4) 算法分析

- 此次实验中在拉普拉斯平滑参数分别取0-5的情况下，平均出错数据约为1500条，准确度约为84%
- 经过统计发现，类似于4、7、9、8等图形较为复杂的数据出错频率较高
- 可能是由于这些图形图像较为复杂，特征值较为集中且易于混淆导致算法出错
- 分析算法的优缺点如下：
- MNIST数据集被预先处理过。通过上面的示例图片可以看出，MNIST中的图片较为纯净，没有噪声干扰，非常清晰。所以实验中可以直接进行二值化。
- MNIST数据集中，图片中的数字总是在中心位置，大小合适，比较饱满。这一点保留了部分的空间信息。
- 误差可能的产生原因之一也有可能是在核心分类器函数中：

  1) 计算样本属于每个类的概率，在第二节中已经说明，只需求式中分子的值即可：
  $P(c_i|X=\{x_1,x_2,\ldots,x_n\}) = log(P(c_i)) + \sum_{j=1}^{n} log(P(x_j|c_i))$
-

  为什么要取对数？对每个 $P(x_j|c_i)$ 的计算结果有的非常接近0，由于计算机精度限制，多个接近0的数相乘，不利于大小的对比。
- 许多接近于0的数字相乘，由于计算机精度的限制，导致大小比对时出现误差

源代码

```
import numpy as np    #计算数据
import matplotlib.pyplot as plt  #绘制图线
import pandas as pd  #读取csv文件 处理DataFrame

def getOriData():
    #读取原始CSV数据 未经处理
    train = pd.read_csv('mnist_train.csv',header=None)  #没有表头 header指定为None
    test = pd.read_csv('mnist_test.csv',header = None)
    return train,test

#二值化映射函数
def Binarization(val):
    tag = 255/2
    if val < tag:
        return 0
    else:
        return 1

def DataProcess(train,test):
```

```
    for i in range(train.shape[0]):
        #二维数组的切片不易理解 转换为一维切片更易理解
        train.loc[i][1:]=train.loc[i][1:].apply(Binarization)
    for i in range(test.shape[0]):
        #二维数组的切片不易理解 转换为一维切片更易理解
        test.loc[i][1:]=test.loc[i][1:].apply(Binarization)
    #拆分标签和特征向量
    label = trainData.iloc[:,0]
    feature = trainData.iloc[:,1:]
    return label,feature

def getPCi(data):
    #计算先验概率 P(C|J)
    tag=0
    counts = [0]*10
    for i in range(data.shape[0]):
        tag=data.loc[i][0]
        counts[tag]+=1
    PCi=np.zeros(10)  #先验概率
    for i in range(10):
        PCi[i] = counts[i]/60000
    return counts,PCi

def TrainModel(df_feature,df_label,counts,k,n,labelNum):
    """
    Parameters:
    df_feature: 特征向量      类型:DataFrame  二值特征 0，1
    df_label: 标签向量        类型:DataFrame 其中值取0-9
    k:拉普拉斯平滑参数      类型:int
    counts:各个类别的样本数量 类型:array
    N:特征种数  本题为2
    labelNum:类别种数         类型:int
    """
    vecc = []   #存放的是每一个的P(Xj|Ci) 10*784
    for i in range(labelNum):
        print("当前正在训练的类别为:手写数字{}类".format(i))
        print("计算I(xj=1,c{})".format(i))
        I = feature[df_label==i].apply(np.sum,axis=0)  #获得每一个特征列中样本特征取值为1的个数
        p=(I+k) / (counts[i]+k*n)  #运用拉普拉斯平滑 计算P(Xj|Ci)
        vecc.append(p)
        print("I(xj=1,c{})={}".format(i,vecc[i][1]))
        print("手写数字{}类训练完成".format(i))
    return vecc


def ClassifySamples(sample,PCi,VEC_P):
    """
    Parameters:
    sample: 训练样本 10*784    类型:二维ndarray
    PCJ: 先验概率   10    类型:float64
    VEC_P: 即E矩阵
    """
    step1 = np.subtract(sample,1)  #数组数据相减
    step2 = np.abs(step1)  #求出绝对值
    step3 = np.subtract(step2,VEC_P)  #再相减
    step4=np.abs(step3) #求出最后的值
    log_P_Ci = np.log(PCi)
    log_P_Xj_Ci = np.log(step4)
    #计算P(Ci|x)=log(P(Ci))+Σlog(P(Xj|Ci))
    P_Ci_X = np.add(log_P_Ci,np.sum(log_P_Xj_Ci,axis=1))
```

```
    return np.argmax(P_Ci_X)


def ClassifyTestData(TestData,PCi,VEC_P):
    temp = []
    correctNum = 0
    FalseNum = 0
    for i in range(TestData.shape[0]):
        sample = TestData.iloc[i,1:].values
        MyAns = ClassifySamples(sample,PCi,VEC_P)
        Ans = TestData.loc[i][0]
        if i%5 == 0:
            print("Ans={},MyAns={}".format(Ans,MyAns))
        if MyAns == Ans:
            correctNum += 1
        else:
            FalseNum += 1
    print("CorrectNum={}".format(correctNum))
    print("FalseNum={}".format(FalseNum))
    correctRate = correctNum / (correctNum+FalseNum)
    return correctRate

trainData,testData = getOriData()
counts,PCi=getPCi(trainData)
label,feature=DataProcess(trainData,testData)
vec = TrainModel(feature,label,counts,3,2,10)

#选取不同的拉普拉斯平滑参数进行测试
def RunNaiveBayes(k):
    trainData,testData = getOriData()
    counts,PCi=getPCi(trainData)
    label,feature=DataProcess(trainData,testData)
    vec = TrainModel(feature,label,counts,k,2,10)
    Rate=ClassifyTestData(testData,PCi,vec)
    return Rate

r1=RunNaiveBayes(1)
r2=RunNaiveBayes(2)
r3=RunNaiveBayes(3)
r4=RunNaiveBayes(4)
r5=RunNaiveBayes(5)

print("k=0,rate={}".format(r0))
print("k=1,rate={}".format(r1))
print("k=2,rate={}".format(r2))
print("k=3,rate={}".format(r3))
print("k=4,rate={}".format(r4))
print("k=5,rate={}".format(r5))
```

实验结果

当前正在训练的类别为:手写数字0类
计算I(xj=1,c0)
I(xj=1,c0)=0.0008427439743805831
手写数字0类训练完成
当前正在训练的类别为:手写数字1类
计算I(xj=1,c1)
I(xj=1,c1)=0.000740521327014218
手写数字1类训练完成
当前正在训练的类别为:手写数字2类
计算I(xj=1,c2)
I(xj=1,c2)=0.0008378016085790885
手写数字2类训练完成
当前正在训练的类别为:手写数字3类
计算I(xj=1,c3)
I(xj=1,c3)=0.0008141996417521576
手写数字3类训练完成
当前正在训练的类别为:手写数字4类
计算I(xj=1,c4)
I(xj=1,c4)=0.0008544087491455912
手写数字4类训练完成
当前正在训练的类别为:手写数字5类
计算I(xj=1,c5)
I(xj=1,c5)=0.0009206407659731173
手写数字5类训练完成
当前正在训练的类别为:手写数字6类
计算I(xj=1,c6)
I(xj=1,c6)=0.0008434547908232119
手写数字6类训练完成
当前正在训练的类别为:手写数字7类
计算I(xj=1,c7)
I(xj=1,c7)=0.0007968127490039841
手写数字7类训练完成
当前正在训练的类别为:手写数字8类
计算I(xj=1,c8)
I(xj=1,c8)=0.0008530967411704487
手写数字8类训练完成
当前正在训练的类别为:手写数字9类
计算I(xj=1,c9)
I(xj=1,c9)=0.0008390669575432119
手写数字9类训练完成

```
Ans=7, MyAns=7
Ans=1, MyAns=1
Ans=0, MyAns=0
Ans=5, MyAns=3
Ans=9, MyAns=9
Ans=0, MyAns=0
Ans=3, MyAns=3
Ans=2, MyAns=2
Ans=1, MyAns=1
Ans=5, MyAns=5
Ans=6, MyAns=6
Ans=0, MyAns=8
Ans=7, MyAns=7
Ans=4, MyAns=9
Ans=7, MyAns=7
Ans=7, MyAns=7
Ans=7, MyAns=9
Ans=4, MyAns=4
  Ans=8, MyAns=8
  Ans=3, MyAns=8
  Ans=0, MyAns=0
  Ans=7, MyAns=7
  Ans=6, MyAns=6
  Ans=9, MyAns=9
  Ans=1, MyAns=1
  Ans=1, MyAns=1
  Ans=4, MyAns=4
  Ans=3, MyAns=3
  Ans=5, MyAns=2
  Ans=3, MyAns=2
  Ans=2, MyAns=2
  Ans=2, MyAns=8
  Ans=7, MyAns=7
  Ans=2, MyAns=2
  CorrectNum=8412
  FalseNum=1588
```

]:
```
print("k=0,rate={}".format(r0))
print("k=1,rate={}".format(r1))
print("k=2,rate={}".format(r2))
print("k=3,rate={}".format(r3))
print("k=4,rate={}".format(r4))
print("k=5,rate={}".format(r5))
```

```
k=0, rate=0.8434
k=1, rate=0.8427
k=2, rate=0.8425
k=3, rate=0.8419
k=4, rate=0.8417
k=5, rate=0.8412
```