模式识别实验报告

专业：	人工智能
学号：	58119104
年级：	2019 级
姓名：	蔡雅清

签名：蔡雅清

时间： 2021 年 6 月 16 日

实验一 贝叶斯分类任务

1. 问题描述

本次实验的任务是利用贝叶斯分类算法对 wine 数据集中的测试集进行分类。

根据实验提供信息，数据集已被划分为训练集和测试集，分别存储于 train_data.csv 和 test_data.csv。其中，训练集包含 120 个样例，测试集包含 58 个样例，每个样例包含各个维度的特征值及样例标签（标签为 1、2 或 3），假定各维度的特征属性之间条件独立。

本次实验需要基于贝叶斯分类原理，实现一个贝叶斯分类器。在训练集中进行训练，尽可能提高模型准确率，并在测试集上进行测试。在本次实验的朴素贝叶斯分类模型中，我们假设连续变量服从高斯分布，并且利用训练集中数据估计高斯分布的均值和方差并用高斯分布来估计类条件概率。

2. 实现步骤与流程

（1） 训练集处理

将.csv文件转换为列表存储，便于后续实验中使用。

（2） 贝叶斯模型过程推导

朴素贝叶斯分类器与贝叶斯分类器不同，它假设所有的输入的分布都是条件独立的。即：

$$P(X_1, X_2, \times\times\times, X_p|C) = P(X_1|C)P(X_2|C)\cdots\cdots P(X_p|C)$$

根据贝叶斯公式我们可以得知：

$$\hat{P}(c_j) = \frac{N(C = c_j)}{N}$$

$$\hat{P}(x_i|c_j) = \frac{N(X_i = x_i, C = c_j)}{N(C = c_j)}$$

朴素贝叶斯算法流程如下：

输入：训练数据$T = \{(x_1, y_1), (x_2, y_2), \ldots\ldots (x_N, y_N)\}$，其中$x_i = (x_i^{(1)}, x_i^{(2)}, \ldots\ldots, x_i^{(n)})^T$，$x_i^{(j)}$是第$i$个样本的第$j$个特征，$x_i^{(j)} \epsilon \{a_{j1}, a_{j2}, \ldots\ldots, a_{jSj}\}$，$a_{jl}$是第$j$个特征可能取的第$l$个值，$j =$

$1,2,\ldots\ldots,n$ ，$l=1,2,\ldots\ldots S_j$；实例x

输出：实例x的分类

a. 计算先验概率及条件概率

$$P(Y=c_k)=\frac{\sum_{i=1}^{N}I(y_i=c_k)}{N},k=1,2,\ldots\ldots K$$

$$P\left(X^{(j)}=a_{jl}|Y=c_k\right)=\frac{\sum_{i=1}^{N}I(x_i{}^{(j)}=a_{jl},y_i=c_k)}{\sum_{i=1}^{N}I(y_i=c_k)}$$

$$j=1,2,\ldots\ldots,n;l=1,2,\ldots\ldots S_j;\ k=1,2,\ldots\ldots K$$

b. 对于给定的实例$x=(x^{(1)},x^{(2)},\ldots\ldots,x^{(n)})^T$，计算

$$P(Y=c_k)\prod_{j=1}^{n}P\left(X^{(j)}=x^{(j)}|Y=c_k\right),k=1,2,\ldots\ldots K$$

c. 确定实例的类

$$y=\arg\max P(Y=c_k)\prod_{j=1}^{n}P\left(X^{(j)}=x^{(j)}|Y=c_k\right)$$

（3） 朴素贝叶斯分类器实现

通过定义$Naive_bayesian_classifier$类，以及数据预处理等部分，实现朴素贝叶斯分类器的功能。其中步骤包括计算先验概率、利用高斯分布估计样本均值和方差、根据给定高斯分布采用朴素贝叶斯分类计算单个样本的类条件概率、计算各个样本在三个类中的后验概率、判断样本所属类别及计算准确率等。

（4） 对 wine 数据集中的测试集进行分类

3. 实验结果与分析

（1） 实验结果展示

```
PS D:\Files\A2020-2021-3\模式识别-薛晖\实验5选3\我的实验\58119104-蔡雅清-贝叶斯分类任务实验>  & 'D:\Softwares\python\python.exe' 'c:\Users\Lenovo\.vscode\extensions\ms-python.python-2021.5.926500501\pythonFiles\lib\python\debugpy\launcher' '18402' '--' 'd:\Files\A2020-2021-3\模式识别-薛晖\实验5选3\我的实验\58119104-蔡雅清-贝叶斯分类任务实验\Classify.py'
          0             1             2             3
0         1  4.590070e-08  1.648065e-22  1.275974e-66
1         1  3.323228e-08  4.973196e-16  1.927058e-50
2         1  1.655039e-07  1.882254e-18  5.784741e-50
3         1  7.153634e-12  6.048613e-37  1.916431e-69
4         1  3.097546e-08  3.457150e-11  2.189682e-33
5         1  2.239395e-08  2.425593e-30  5.370984e-54
6         1  1.126178e-07  8.346939e-24  2.068050e-53
7         1  1.398712e-07  2.282588e-22  1.729650e-52
8         1  1.601489e-08  1.800615e-22  1.453777e-42
9         1  7.030520e-07  1.641528e-22  3.060484e-55
10        1  1.698873e-08  1.905161e-28  2.483109e-63
11        1  4.131363e-08  9.485300e-21  1.230305e-37
12        1  8.057472e-07  3.101714e-21  6.004212e-43
13        1  1.493694e-13  3.262075e-27  2.338103e-59
14        1  2.713150e-13  1.837092e-38  5.049007e-71
15        1  7.447361e-08  9.271726e-26  1.660367e-46
16        1  6.604232e-08  4.672382e-25  1.498781e-41
17        1  6.769190e-08  1.992068e-21  4.740680e-41
18        1  4.964937e-13  5.235960e-43  3.771231e-71
19        2  2.445262e-35  2.048742e-16  4.046972e-29
20        2  3.674721e-29  6.461446e-13  1.524662e-19
21        3  2.780251e-24  4.510169e-14  1.105981e-13
22        2  6.538738e-15  1.116633e-11  9.955756e-25
23        2  1.908087e-13  3.344309e-10  1.831310e-46
24        2  9.408949e-23  3.842778e-09  7.748735e-28
25        2  2.457114e-11  2.967130e-09  2.432153e-27
26        1  2.784523e-12  2.483216e-13  8.598341e-53
27        2  4.475030e-16  2.407424e-09  3.302904e-42
28        2  2.037577e-20  2.775498e-13  1.056977e-15
29        2  1.968700e-24  3.311782e-14  1.115972e-43
30        2  2.057007e-25  1.418097e-11  2.178881e-12
31        2  1.288799e-13  1.050916e-12  1.505122e-49
32        2  2.036079e-15  8.209031e-10  1.218762e-22
33        1  6.014220e-17  8.565952e-18  2.698412e-63
34        2  1.483973e-13  1.031608e-10  1.871914e-38
35        2  4.325466e-24  1.000216e-09  2.753989e-23
36        2  3.268097e-16  4.756031e-11  3.848436e-29
37        2  9.689621e-23  1.751035e-08  2.696419e-18
38        2  1.666495e-18  2.405643e-12  6.499979e-27
39        2  2.014885e-13  3.004776e-08  7.702968e-39
40        3  1.775443e-25  2.357611e-14  5.685729e-12
41        3  1.698066e-26  9.825718e-15  1.013896e-08
42        3  9.382076e-31  5.847297e-16  1.107625e-08
43        3  3.129956e-26  1.621327e-14  8.577024e-10
44        3  1.971519e-30  1.643333e-14  8.849448e-10
45        3  4.459022e-32  9.446868e-19  6.509549e-08
46        3  1.503146e-38  1.815172e-15  1.680145e-10
47        3  1.268690e-36  9.364925e-16  8.416605e-10
48        3  3.244275e-30  2.853854e-16  2.151516e-08
49        3  4.528179e-25  3.928548e-12  1.767364e-10
50        3  5.889596e-28  2.761426e-12  9.053593e-11
51        3  1.050194e-25  3.709541e-15  2.617615e-12
52        3  3.688294e-29  3.846248e-14  1.790991e-10
53        3  4.596077e-24  1.126559e-13  4.346742e-10
54        3  2.373222e-28  2.957253e-22  1.396472e-09
55        3  7.152884e-29  1.728795e-14  3.204100e-08
56        3  7.806272e-43  5.948221e-21  5.640794e-12
57        3  9.074796e-33  1.175018e-20  5.416341e-07
58  预测正确率  9.482759e-01           NaN           NaN
PS D:\Files\A2020-2021-3\模式识别-薛晖\实验5选3\我的实验\58119104-蔡雅清-贝叶斯分类任务实验>
```

（2） 分析

由实验结果展示可知，本次实验所设计的朴素贝叶斯分类器在给定 wine 数据集中的预测正确率为 94.83%，准确率较高。

朴素贝叶斯分类器的算法逻辑简单,易于实现只要使用贝叶斯公式转化一下即可，且由于假设特征相互独立，故分类过程中时空开销小只会涉及到二维存储。

理论上，朴素贝叶斯模型与其他分类方法相比具有最小的误差率。但是实际上并非总是如此，这是因为朴素贝叶斯模型假设属性之间相互独立，这个假设在实际应用中往往是不成立的，在属性个数比较多或者属性之间相关性较大时，分类效果不好。而在属性相关性较小时，朴素贝叶斯性能最为良好。对于这一点，有半朴素贝叶斯之类的算法通过考虑部分关联性适度改

进。

4. 代码附录

Baysesian_Classifier.py

```
# coding:utf-8


import numpy as np
import pandas as pd
import math

train_data = pd.read_csv(r"train_data.csv",header=None)
# print(train_data)

# 将dataframe转换为list
train_datasets=train_data.values.tolist()
# print("list",train_data)

# 计算各个类的总数以及训练样本总数
def count_total(data):
    count = [0,0,0]
    total = 0
    training_data=np.array(data)
    classes=training_data[:,0]
    for num in classes:
        # 统计各个类的总数
        if num == 1: count[0] += 1
        if num == 2: count[1] += 1
        if num == 3: count[2] += 1
    total = count[0]+count[1]+count[2]
    return count, total
# print("total",count_total(train_datasets))


# 计算各类红酒的先验概率
def cal_prior_rate(data):
    classes, total = count_total(data)
    cal_prior_rate = [0,0,0]
    for label in range(3):
        prior_prob = classes[label] / total
        cal_prior_rate[label] = prior_prob
    return cal_prior_rate
# print("prior",cal_prior_rate(train_datasets))
```

```
# 利用高斯分布估计均值和方差
# 本实验规定采用高斯分布估计类条件概率。其中，均值和方差分别用训练集的样本均值和样本方差估计。
def cal_Gaussian(data):
    count, total = count_total(data)
    new_data=np.array(data)
    all_model = []
    for i in range(1,14,1):
        # 遍历十三个特征值
        feature=new_data[:,i]
        para=[]
        model = []
        sum1 = 0
        sum2 = 0
        for k in range(count[0]):
            # 遍历第一个类中各个样本的第 i 个特征
            sum1 += feature[k]
        mean=sum1/count[0]
        model.append(mean)
        for k in range(count[0]):
            sum2 += (feature[k]-mean)**2
        variance=sum2/count[0]
        model.append(variance)
        para.append(model)
        model = []
        sum1 = 0
        sum2 = 0
        for k in range(count[0],count[0]+count[1]):
            # 遍历第二个类中各个样本的第 i 个特征
            sum1 += feature[k]
        mean=sum1/count[1]
        model.append(mean)
        for k in range(count[0], count[0] + count[1]):
            sum2 += (feature[k]-mean)**2
        variance=sum2/count[1]
        model.append(variance)
        para.append(model)
        model = []
        sum1 = 0
        sum2 = 0
        for k in range(count[0]+count[1],total):
            # 遍历第三个类中各个样本的第 i 个特征
            sum1 += feature[k]
        mean=sum1/count[2]
```

```
        model.append(mean)
        for k in range(count[0] + count[1], total):
            sum2 += (feature[k]-mean)**2
        variance=sum2/count[2]
        model.append(variance)
        para.append(model)
        all_model.append(para)
    return all_model
# print("Gaussian",cal_Gaussian(train_datasets))


def cal_likelihood(data):
    '''给定高斯分布采用朴素贝叶斯分类计算单个样本的类条件概率'''
    model=cal_Gaussian(train_datasets)
    a=np.array(model)
    likelihood=[]
    multi=1
    feature_model = a[:, 0, :]#取13 个特征值的第一类高斯分布模型的参数
    for i in range(13):#遍历每个特征，计算该样本的类条件概率
        part_model=feature_model[i]
        prob=1/(math.sqrt(2*math.pi*part_model[1]))*\
             math.exp((-(data[i+1]-
part_model[0])**2)/(2*part_model[1]))
        multi *= prob
    likelihood.append(multi)
    multi=1
    feature_model = a[:, 1, :] # 取13 个特征值的第二类高斯分布模型的参数
    for i in range(13): # 遍历每个特征，计算该样本的类条件概率
        part_model=feature_model[i]
        prob=1/(math.sqrt(2*math.pi*part_model[1]))*\
            math.exp((-(data[i+1]-part_model[0])**2)/(2*part_model[1]))
        multi *= prob
    likelihood.append(multi)
    multi=1
    feature_model = a[:, 2, :]
    # 取13 个特征值的第三类高斯分布模型的参数
    for i in range(13):
        # 遍历每个特征，计算该样本的类条件概率
        part_model=feature_model[i]
        prob = 1/(math.sqrt(2*math.pi*part_model[1]))*\
            math.exp((-(data[i+1]-part_model[0])**2)/(2*part_model[1]))
        multi *= prob
    likelihood.append(multi)
    return likelihood
# print(train_datasets[0])
```

```
# print("likelihood",cal_likelihood(train_datasets[0]))

class Naive_bayesian_classifier:
    def __init__(self, data):
        self._data = data # 初始化数据
        self._model=cal_Gaussian(train_datasets) # 用训练数据初始化高斯分布的模型
        a=np.array(self._data)
        self._initial_labels = a[:,0]  # 存放最初的类别标签
        self._final_labels=[] # 初始化最终类别标签[,...,]
        self._prior_prob = cal_prior_rate(train_datasets) # 计算各个类的先验概率
        self._likelihood_prob = [] # 计算各个样本的在三个类的类条件概率[[,,]...[,,]]
        self._post_prob = [] # 存放各个样本在三个类的后验概率[[,,]...[,,]]
        self._correct_rate=0 # 正确率

    # 下面的函数可以直接调用上面类中定义的变量
    def get_likelihood_prob(self):
        # 计算各个样本在三个类的类条件概率
        for sample in self._data:
            self._likelihood_prob.append(cal_likelihood(sample))

    def get_post_prob(self):
        # 计算各个样本在三个类中的后验概率
        for sample in self._likelihood_prob:
            post=[]
            for i in range(3):
                post.append(sample[i]*self._prior_prob[i])
            self._post_prob.append(post)
        return self._post_prob

    def get_final_labels(self):
        # 判断各个样本最终类别
        for sample in self._post_prob:
            max=sample[0]
            label=1
            if sample[1]>max:
                max=sample[1]
                label=2
            if sample[2]>max:
                max=sample[2]
                label=3
            self._final_labels.append(label)
```

```
        return self._final_labels

    def get_correct_rate(self):
        # 计算准确率
        count, total = count_total(self._data)
        sum = 0
        for i in range(total):
            if self._initial_labels[i]==self._final_labels[i]:
                sum += 1
        self._correct_rate=float(sum/total)
        return self._correct_rate
```

Classify.py

```
# 分类
# coding:utf-8
import pandas as pd
import Bayesian_Classifier


def main():
    test_data=pd.read_csv(r"test_data.csv",header=None)
    test_datasets=test_data.values.tolist()
    classfier = Bayesian_Classifier.Naive_bayesian_classifier(test_data
sets)
    count,total = Bayesian_Classifier.count_total(test_datasets)
    classfier.get_likelihood_prob() # 计算各个样本在三个类的类条件概率
    post=classfier.get_post_prob() # 计算各个样本在三个类中的后验概率
    #print(post)
    labels=classfier.get_final_labels() # 判断各个样本最终类别
    result=[]
    for i in range(total):
        # 遍历样本
        sample=[]
        sample.append(labels[i])
        for j in post[i]:
            sample.append(j)
        result.append(sample)
    result.append(['预测正确率',classfier.get_correct_rate()])
    test = pd.DataFrame(data=result)
    print(test)
    test.to_csv('D:/Files/A2020-2021-3/模式识别-薛晖/实验 5 选 3/我的实验
/58119104-蔡雅清-贝叶斯分类任务实验/test_prediction.csv',
                encoding='gbk', header=None , index=None)
```

```python
if __name__ == '__main__':
    main()
```

实验二 隐马尔可夫模型分词任务

1. 问题描述

在人民日报分词语料库上统计语料信息，对隐马尔科夫模型进行训练。利用训练好的模型，对以下语句进行分词测试：

1) 今天我来到了东南大学。

2) 模式识别课程是一门有趣的课程。

3) 我认为完成本次实验是一个挑战。

2. 实现步骤与流程

（1） 训练HMM模型

定义函数$Init_Array()$初始化所有概率矩阵；

定义$get_tag(word)$对训练集获取状态标签；

定义$Prob_Array()$将参数估计的概率取对数，对概率 0 取无穷小；

定义$Dic_Array(array_b)$将字典转换为数组；

定义$dist_tag()$用于判断一个字最大发射概率的状态。

一个HMM可以由下面几个部分组成：（1）模型中的状态数量 N（上例中盒子的数量）在中文分词中一般只有 4 个状态：$STATES = \{ ‘B’, ‘M’, ‘E’, ‘S’ \}$. 从每个状态可能输出的不同符号的数目 M 在中文分词中就是对应每一个字符.

（2） 利用$Viterbi$算法预测

定义$Viterbi$算法对应函数$Viterbi()$求测试集最优状态序列。

从语料库中训练HMM 分词模型后，可通过$Viterbi$算法来预测未知语言中汉字的词位标记从而达到分词的目的，可以求得全局最优的分词结果。$Viterbi$ 算法实际是用动态规划来求解隐马尔可夫模型预测问题，即用动态规划求概率最大路径。

（3） 分词

定义分词函数$tag_seg()$根据状态序列进行分词。

分词的方法是从左到右，采用最大匹配模式。

3. 实验结果与分析

实验结果如下：

```
PS D:\Files\A2020-2021-3\模式识别-薛晖\实验5选3\我的实验\5811910
ensions\ms-python.python-2021.5.842923320\pythonFiles\lib\python
清-隐马尔科夫模型分词任务实验\main.py'
今天 我来 到 了 东南 大学 。
模式 识别 课程 是 一门 有趣 的 课程 。
我 认为 完成 本次 实验 是 一个 挑战 。
```

分析：

（1） 实验结果表明本次实验利用隐马尔可夫模型以及人民日报语料库训练出的分词模型能够较好地完成所给出的任务；

（2） 但在第一句话中“我来”被分在了一起，这是因为基于HMM的分词模型经常会将一起出现频率高的字组切分成词发的情况，如“我的”、“每个”等，出现错误分词的现象。

（3） HMM模型在有训练语料时，训练模型的时间较短。HMM 是生成模型，即使没有先验语料，也可以使用EM方法进行估计，估计原则是使每个序列的 P(X)最大，这个优势是判别模型无法比拟的。

（4） HMM分词模型只考虑词前后关系，未考虑词的上下文之间的关系，但在中文分词中表现较好，HMM可以求得全局最优的分词结果。中文分词涉及的范围非常广，由于中文本身的特殊性，中文分词算法在不断地发展和完善，在分词速度更快、精度更高、歧义词、未登录词、新词的识别等方面会得到突破。

4. 代码附录

HMM.py

```python
#coding:utf-8

import pandas as pd
import codecs
from numpy import *
import numpy as np
import sys
import re

STATES = ['B', 'M', 'E', 'S']
array_A = {}    #状态转移概率矩阵
```

```
array_B = {}     #发射概率矩阵
array_E = {}     #测试集存在的字符，但在训练集中不存在，发射概率矩阵
array_Pi = {}    #初始状态分布
word_set = set()     #训练数据集中所有字的集合
count_dic = {}  #‘B,M,E,S’每个状态在训练集中出现的次数
line_num = 0     #训练集语句数量

#初始化所有概率矩阵
def Init_Array():
    for state0 in STATES:
        array_A[state0] = {}
        for state1 in STATES:
            array_A[state0][state1] = 0.0
    for state in STATES:
        array_Pi[state] = 0.0
        array_B[state] = {}
        array_E = {}
        count_dic[state] = 0

#对训练集获取状态标签
def get_tag(word):
    tag = []
    if len(word) == 1:
        tag = ['S']
    elif len(word) == 2:
        tag = ['B', 'E']
    else:
        num = len(word) - 2
        tag.append('B')
        tag.extend(['M'] * num)
        tag.append('E')
    return tag


#将参数估计的概率取对数，对概率 0 取无穷小-3.14e+100
def Prob_Array():
    for key in array_Pi:
        if array_Pi[key] == 0:
            array_Pi[key] = -3.14e+100
        else:
            array_Pi[key] = log(array_Pi[key] / line_num)
    for key0 in array_A:
        for key1 in array_A[key0]:
            if array_A[key0][key1] == 0.0:
```

```
                array_A[key0][key1] = -3.14e+100
            else:
                array_A[key0][key1] = log(array_A[key0][key1] / count_d
ic[key0])
    # print(array_A)
    for key in array_B:
        for word in array_B[key]:
            if array_B[key][word] == 0.0:
                array_B[key][word] = -3.14e+100
            else:
                array_B[key][word] = log(array_B[key][word] /count_dic[
key])

#将字典转换成数组
def Dic_Array(array_b):
    tmp = np.empty((4,len(array_b['B'])))
    for i in range(4):
        for j in range(len(array_b['B'])):
            tmp[i][j] = array_b[STATES[i]][list(word_set)[j]]
    return tmp

#判断一个字最大发射概率的状态
def dist_tag():
    array_E['B']['begin'] = 0
    array_E['M']['begin'] = -3.14e+100
    array_E['E']['begin'] = -3.14e+100
    array_E['S']['begin'] = -3.14e+100
    array_E['B']['end'] = -3.14e+100
    array_E['M']['end'] = -3.14e+100
    array_E['E']['end'] = 0
    array_E['S']['end'] = -3.14e+100

def dist_word(word0,word1,word2,array_b):
    if dist_tag(word0,array_b) == 'S':
        array_E['B'][word1] = 0
        array_E['M'][word1] = -3.14e+100
        array_E['E'][word1] = -3.14e+100
        array_E['S'][word1] = -3.14e+100
    return
```

Viterbi.py

```python
#coding: utf:8

import HMM
#from HMM import STATES

#Viterbi 算法求测试集最优状态序列
def Viterbi(sentence,array_pi,array_a,array_b):
    tab = [{}]  #动态规划表
    path = {}

    if sentence[0] not in array_b['B']:
        for state in HMM.STATES:
            if state == 'S':
                array_b[state][sentence[0]] = 0
            else:
                array_b[state][sentence[0]] = -3.14e+100

    for state in HMM.STATES:
        tab[0][state] = array_pi[state] + array_b[state][sentence[0]]
        # print(tab[0][state])
        #tab[t][state]表示时刻t 到达 state 状态的所有路径中，概率最大路径的概
率值
        path[state] = [state]
    for i in range(1,len(sentence)):
        tab.append({})
        new_path = {}
        # if sentence[i] not in array_b['B']:
        #     print(sentence[i-1],sentence[i])
        for state in HMM.STATES:
            if state == 'B':
                array_b[state]['begin'] = 0
            else:
                array_b[state]['begin'] = -3.14e+100
        for state in HMM.STATES:
            if state == 'E':
                array_b[state]['end'] = 0
            else:
                array_b[state]['end'] = -3.14e+100
        for state0 in HMM.STATES:
            items = []
            # if sentence[i] not in word_set:
            #     array_b[state0][sentence[i]] = -3.14e+100
            # if sentence[i] not in array_b[state0]:
            #     array_b[state0][sentence[i]] = -3.14e+100
```

```
            # print(sentence[i] + state0)
            # print(array_b[state0][sentence[i]])
            for state1 in HMM.STATES:
                # if tab[i-1][state1] == -3.14e+100:
                #     continue
                # else:
                if sentence[i] not in array_b[state0]:  #所有在测试集出现
但没有在训练集中出现的字符
                    if sentence[i-1] not in array_b[state0]:
                        prob = tab[i - 1][state1] + array_a[state1][sta
te0] + array_b[state0]['end']
                    else:
                        prob = tab[i - 1][state1] + array_a[state1][sta
te0] + array_b[state0]['begin']
                    # print(sentence[i])
                    # prob = tab[i-
1][state1] + array_a[state1][state0] + array_b[state0]['other']
                else:
                    prob = tab[i-
1][state1] + array_a[state1][state0] + array_b[state0][sentence[i]]
#计算每个字符对应STATES的概率
#                     print(prob)
                items.append((prob,state1))
            # print(sentence[i] + state0)
            # print(array_b[state0][sentence[i]])
            # print(sentence[i])
            # print(items)
            best = max(items)   #bset:(prob,state)
            # print(best)
            tab[i][state0] = best[0]
            # print(tab[i][state0])
            new_path[state0] = path[best[1]] + [state0]
        path = new_path

    prob, state = max([(tab[len(sentence) - 1][state], state) for state
 in HMM.STATES])
    return path[state]
```

wordSplit.py

```
#coding: utf:8
import HMM
import Viterbi
```

```
#根据状态序列进行分词
def tag_seg(sentence,tag):
    word_list = []
    start = -1
    started = False

    if len(tag) != len(sentence):
        return None

    if len(tag) == 1:
        word_list.append(sentence[0])   #语句只有一个字，直接输出

    else:
        if tag[-1] == 'B' or tag[-1] == 'M':    #最后一个字状态不是'S'或
'E'则修改
            if tag[-2] == 'B' or tag[-2] == 'M':
                tag[-1] = 'E'
            else:
                tag[-1] = 'S'


        for i in range(len(tag)):
            if tag[i] == 'S':
                if started:
                    started = False
                    word_list.append(sentence[start:i])
                word_list.append(sentence[i])
            elif tag[i] == 'B':
                if started:
                    word_list.append(sentence[start:i])
                start = i
                started = True
            elif tag[i] == 'E':
                started = False
                word = sentence[start:i + 1]
                word_list.append(word)
            elif tag[i] == 'M':
                continue

    return word_list
```

main.py

```
#coding: utf:8
```

```
import HMM
import Viterbi
import wordSplit


if __name__ == '__main__':
    trainingSet = open('RenMinData.txt_utf8', encoding='utf-8')      #读
取训练集
    testSet = open('testSet.txt', encoding='utf-8')        #读取测试集
    # trainlist = []

    HMM.Init_Array()

    for line in trainingSet:
        line = line.strip()
        # trainlist.append(line)
        HMM.line_num += 1

        word_list = []
        for k in range(len(line)):
            if line[k] == ' ':continue
            word_list.append(line[k])
        # print(word_list)
        word_set = HMM.word_set | set(word_list)    #训练集所有字的集合

        line = line.split(' ')
        # print(line)
        line_state = []     #这句话的状态序列

        for i in line:
            line_state.extend(HMM.get_tag(i))
        # print(line_state)
        HMM.array_Pi[line_state[0]] += 1  # array_Pi 用于计算初始状态分布
概率


        for j in range(len(line_state)-1):
            # count_dic[line_state[j]] += 1   #记录每一个状态的出现次数
            HMM.array_A[line_state[j]][line_state[j+1]] += 1  #array_A
计算状态转移概率


        for p in range(len(line_state)):
            HMM.count_dic[line_state[p]] += 1  # 记录每一个状态的出现次数
            for state in HMM.STATES:
                if word_list[p] not in HMM.array_B[state]:
```

```
                    HMM.array_B[state][word_list[p]] = 0.0  #保证每个字都
在STATES的字典中
            # if word_list[p] not in array_B[line_state[p]]:
            #     # print(word_list[p])
            #     array_B[line_state[p]][word_list[p]] = 0
            # else:
            HMM.array_B[line_state[p]][word_list[p]] += 1  # array_B用
于计算发射概率

    HMM.Prob_Array()    #对概率取对数保证精度

    output = ''

    for line in testSet:
        line = line.strip()
        tag = Viterbi.Viterbi(line, HMM.array_Pi, HMM.array_A, HMM.arra
y_B)
        # print(tag)
        seg = wordSplit.tag_seg(line, tag)
        # print(seg)
        list = ''
        for i in range(len(seg)):
            list = list + seg[i] + ' '
        # print(list)
        output = output + list + '\n'
    print(output)
    outputfile = open('output.txt', mode='w', encoding='utf-8')
    outputfile.write(output)
```

实验三 KNN 分类任务

1. 问题描述

本实验需要利用 KNN 算法对输血服务中心数据集中的测试集进行分类。

共需完成以下两个任务：

任务一：利用欧式距离、切比雪夫距离、曼哈顿距离作为 KNN 算法的度量函数对测试集进行分类。

任务二：利用马氏距离作为 KNN 算法的度量函数，对测试集进行分类。马氏距离是一种可学习的度量函数，定义如下：

$$d_M(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sqrt{(\boldsymbol{x}_i - \boldsymbol{x}_j)^T \boldsymbol{M}(\boldsymbol{x}_i - \boldsymbol{x}_j)}$$

其中，$\boldsymbol{M} \in \mathbb{R}^{d \times d}$是一个半正定矩阵，是可以学习的参数。由于 的半正定性质，可将上述定义表述为：

$$d_M(x_i, x_j) = \sqrt{(x_i - x_j)^T A^T A (x_i - x_j)} = \|Ax_i - Ax_j\|_2$$

其中，矩阵$\boldsymbol{A} \in \mathbb{R}^{e \times d}$。故马氏距离可以理解为对原始特征进行线性映射，然后计算欧式距离。

给定以下目标函数，在训练集上利用梯度下降法对马氏距离进行学习：

$$\max_{\boldsymbol{A}} f(\boldsymbol{A}) = \max_{\boldsymbol{A}} \sum_{i=1}^{N} \sum_{j \in c_i} p_{ij}$$

其中，C_i表示与样例x_i同类的样例集合，p_{ij}定义为：

$$p_{ij} = \begin{cases} \dfrac{\exp\left(-d_M(x_i, x_j)^2\right)}{\sum_{k \neq i} \exp\left(-d_M(x_i, x_k)^2\right)} & j \neq i \\ 0 & j = i \end{cases}$$

2. 实现步骤与流程

（一） 任务一

1. 三种距离度量计算方法

（1） 欧氏距离

$$D(a,b)=(\sum_{k=1}^{d}(a_k-b_k)^2)^{\frac{1}{2}}$$

欧氏距离对于尺度的缩放不具有鲁棒性

（2） 切比雪夫距离

$$D_{12}=\lim_{k\to\infty}(\sum_{i=1}^{n}|x_{1i}-x_{2i}|^k)$$

（3） 曼哈顿距离

$$D_{12}=\sum_{k=1}^{n}|x_{1k}-x_{2k}|$$

曼哈顿距离也即L_1范数

2. 分别利用三种距离对测试集进行分类

（二） 任务二

1. 马氏距离度量计算方法

$$d_M(x_i,x_j)=\sqrt{(x_i-x_j)^T-M(x_i-x_j)}$$

由M的半正定性可以得到

$$d_M(x_i,x_j)=\sqrt{(x_i-x_j)^TA^TA(x_i-x_j)}=||Ax_i-Ax_j||_2$$

根据马氏距离度量计算方法定义计算马氏距离的函数

2. 定义 UPA 类利用梯度更新方法更新马氏距离的可学习参数 A
3. 根据目标函数，在训练集上利用梯度下降法对马氏距离进行学习。目标函数如下：

$$\max_{\boldsymbol{A}} f(\boldsymbol{A})=\max_{\boldsymbol{A}}\sum_{i=1}^{N}\sum_{j\in C_i}p_{ij}$$

其中C_i表示与样例x_i同类的样例集合，p_{ij}定义为：

$$p_{ij}=\begin{cases}\dfrac{exp\left(-d_M(\boldsymbol{x}_i,\boldsymbol{x}_j)^2\right)}{\sum_{k\neq i}exp(-d_M(\boldsymbol{x}_i,\boldsymbol{x}_k)^2)} & j\neq i\\ 0 & j=i\end{cases}$$

4. 推导过程如下：

$$p_i = \sum_{j \in C_i} p_{ij}$$

$$C_i = \{j | y_j = y_i\}$$

$$f(A) = \sum_i p_{ij}$$

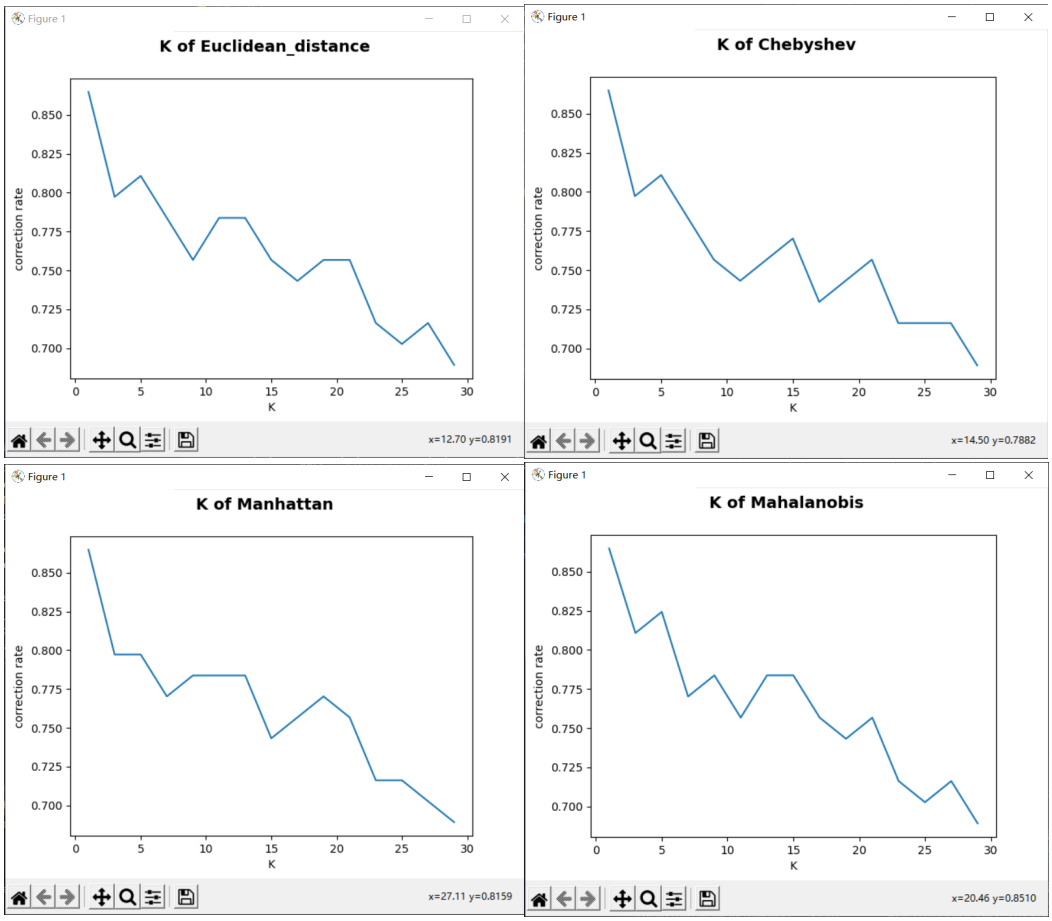
$$\frac{\partial f}{\partial A} = 2A \sum_i (p_i \sum_k p_{ik} x_{ik} x_{ik}{}^T - \sum_{j \in C_i} p_{ij} x_{ij} x_{ij}{}^T)$$

5. 利用马氏距离对测试集进行分类

3. 实验结果与分析

（1）分类结果分别存储在 task1_test_Euclidean.csv，task1_test_Chebyshev.csv，task1_test_Manhattan.csv，task2_test_prediction.csv 中。

（2）由 KNN 原理我们可以知道当 K 太小时会造成过拟合现象，而当 K 太小时会造成欠拟合现象，所以我们应该通过测试一定范围的 K 从而找出能做出最好决策的 KNN 模型，以下为四种不同度量方式不同的 K 对应的准确度：



通过上述结果我们可以看出 K=1 时，分类的准确率均能达到 0.9 以上，故选择 K=1 进行 K 值，并利用其来分别得到对应的四种距离度量得出的 KNN 预测序列。

同时通过结果分析和课堂中所学的三种距离度量的不同特性，我们可以明显看出，四种度量方式都展现了很好地效果，但同时也体现了三种距离度量的不同特点：

a. 欧式距离可以很好地度量出低维的两点的最短距离，而本题中恰好只有四个维度，所以欧氏距离可以很好的反映出数据之间的联系，从而达到较好的分类效果

b. 曼哈顿距离并不能度量出数据间的最短距离，但曼哈顿距离可以比较好地适用于高维数据，因为其计算复杂度相比于欧氏距离显著降低

c. 切比雪夫距离一般在一些特定的问题上才具有良好的性能如度量期盼距离，其不具有如欧式距离的泛化能力。

d. 马氏距离与量纲无关，能够排除变量间相关性的干扰，同时实验利用梯度下降法对马氏距离进行学习。

4. 代码附录

data_process.py

```
#coding :utf-8

import csv

# 训练集导入
with open('train_data.csv',encoding='utf-8') as train:
    train_csv=csv.reader(train)
    tmp=list(train_csv)
    tmp.remove(tmp[0])
    train_data =[]
    for row in tmp:
        trow=[]
        for element in row:
            trow.append(float(element))
        train_data.append(trow)
#print(train_data)
```

```
# 验证集导入
with open('val_data.csv',encoding='utf-8') as valid:
    valid_csv=csv.reader(valid)
    temp=list(valid_csv)
    temp.remove(temp[0])
    val_data =[]
    for row in temp:
        trow=[]
        for element in row:
            trow.append(float(element))
        val_data.append(trow)
#print(val_data)

#导入测试集
with open('test_data.csv',encoding='utf-8') as f:
    f_csv=csv.reader(f)
    temp=list(f_csv)
    temp.remove(temp[0])
    test_data =[]
    for row in temp:
        trow=[]
        for element in row:
            trow.append(float(element))
        test_data.append(trow)
#print(test_data)


def count_total(dataset):
    return len(dataset)


print(count_total(train_data))
```

task_1.py

```
#coding :utf-8

'''任务一：利用欧式距离、切比雪夫距离、曼哈顿距离作为 KNN 算法的度量函数对测试集进行分类'''

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

```
import data_process

def Euclidean_distance(train_data,data):
    '''计算单个样本与训练集各个样本间的 Euclidean_distance，并按升序排列'''
    a1=np.array(train_data)
    a2=np.array(data)
    td=a1[:,0:4]
    d=a2[0:4]
    label=a1[:,4]
    E_distance=[]
    dis_lab=[] # 存储该样本与训练样本的欧式距离和训练样本对应的类别
    for sample in td:
        norm=np.linalg.norm(sample-d)
        E_distance.append(norm)
    for i in range(data_process.count_total(train_data)):
        dis_lab.append([E_distance[i],label[i]])
    sort=sorted(dis_lab,key=(lambda x:x[0])) # 将欧氏距离按升序排列
    return sort
#print(Euclidean_distance(data_process.train_data,data_process.val_data
[0]))

def Manhattan_distance(train_data,data):
    '''计算单个样本与训练集各个样本间的 Manhattan_distance，并按升序排列'''
    a1=np.array(train_data)
    a2=np.array(data)
    td=a1[:,0:4]
    d=a2[0:4]
    label=a1[:,4]
    C_distance=[]
    dis_lab=[]#存储该样本与训练样本的 Manhattan_distance 和训练样本对应的类
别
    for sample in td:
        temp=np.array([feature for feature in sample-d])
        norm=sum(abs(temp))
        C_distance.append(norm)
    for i in range(data_process.count_total(train_data)):
        dis_lab.append([C_distance[i],label[i]])
    sort=sorted(dis_lab,key=(lambda x:x[0]))#将 Manhattan_distance 按升序
排列
    return sort
#print(Manhattan_distance(data_process.train_data,data_process.val_data
[0]))

def Chebyshev_distance(train_data,data):
```

```
    '''计算单个样本与训练集各个样本间的 Chebyshev_distance，并按升序排列'''
    a1=np.array(train_data)
    a2=np.array(data)
    td=a1[:,0:4]
    d=a2[0:4]
    label=a1[:,4]
    C_distance=[]
    dis_lab=[]#存储该样本与训练样本的 Chebyshev_distance 和训练样本对应的类别
    for sample in td:
        norm=max(abs(sample-d))
        C_distance.append(norm)
    for i in range(data_process.count_total(train_data)):
        dis_lab.append([C_distance[i],label[i]])
    sort=sorted(dis_lab,key=(lambda x:x[0]))#将 Chebyshev_distance 按升序排列
    return sort
#print(Chebyshev_distance(data_process.train_data,data_process.val_data[0]))

def decide_label_Euclidean(train_data,val_data):
    '''用验证集计算各个 K 下的正确率'''
    total=data_process.count_total(val_data)
    a=np.array(val_data)
    data=a[:,0:4]
    label=a[:,4]
    corr = []#各个 K 下的正确率
    for K in range(1,30,2):
        predict = []  # 对验证集各样本进行分类
        for sample in data:
            sort = Euclidean_distance(train_data, sample)
            lab_0=0
            lab_1=0
            for i in range(K):
                if sort[i][1]==0:
                    lab_0+=1
                else:
                    lab_1+=1
            if lab_0>lab_1:
                predict.append(0)
            else:
                predict.append(1)
        correct=0
        for i , lab in enumerate(label):
```

```
            if predict[i]==lab:
                correct+=1
        corr.append(correct/total)
    #绘制K 对精度影响的曲线图
    K=list(range(1,30,2))
    fig=plt.figure()
    ax = fig.add_subplot(1, 1, 1)
    fig.suptitle('K of Euclidean_distance', fontsize = 14, fontweight='
bold')
    ax.set_xlabel("K")
    ax.set_ylabel("correction rate")
    plt.plot(K, corr)
    plt.show()
    K1=corr.index(max(corr))*2+1#记录准确率最高的K
    return K1

def decide_label_Chebyshev(train_data,val_data):
    '''用验证集计算各个K下的正确率'''
    total=data_process.count_total(val_data)
    a=np.array(val_data)
    data=a[:,0:4]
    label=a[:,4]
    corr = []#各个K 下的正确率
    for K in range(1,30,2):
        predict = []  # 对验证集各样本进行分类
        for sample in data:
            sort = Chebyshev_distance(train_data, sample)
            lab_0=0
            lab_1=0
            for i in range(K):
                if sort[i][1]==0:
                    lab_0+=1
                else:
                    lab_1+=1
            if lab_0>lab_1:
                predict.append(0)
            else:
                predict.append(1)
        correct=0
        for i , lab in enumerate(label):
            if predict[i]==lab:
                correct+=1
        corr.append(correct/total)
    #绘制K 对精度影响的曲线图
```

```
    K=list(range(1,30,2))
    fig=plt.figure()
    ax = fig.add_subplot(1, 1, 1)
    fig.suptitle('K of Chebyshev', fontsize = 14, fontweight='bold')
    ax.set_xlabel("K")
    ax.set_ylabel("correction rate")
    plt.plot(K, corr)
    plt.show()
    K1=corr.index(max(corr))*2+1#记录准确率最高的K
    return K1

def decide_label_Manhattan(train_data,val_data):
    '''用验证集计算各个K下的正确率'''
    total=data_process.count_total(val_data)
    a=np.array(val_data)
    data=a[:,0:4]
    label=a[:,4]
    corr = []#各个K下的正确率
    for K in range(1,30,2):
        predict = []  # 对验证集各样本进行分类
        for sample in data:
            sort = Manhattan_distance(train_data, sample)
            lab_0=0
            lab_1=0
            for i in range(K):
                if sort[i][1]==0:
                    lab_0+=1
                else:
                    lab_1+=1
            if lab_0>lab_1:
                predict.append(0)
            else:
                predict.append(1)
        correct=0
        for i , lab in enumerate(label):
            if predict[i]==lab:
                correct+=1
        corr.append(correct/total)
    #绘制K对精度影响的曲线图
    K=list(range(1,30,2))
    fig=plt.figure()
    ax = fig.add_subplot(1, 1, 1)
    fig.suptitle('K of Manhattan', fontsize = 14, fontweight='bold')
    ax.set_xlabel("K")
```

```
    ax.set_ylabel("correction rate")
    plt.plot(K, corr)
    plt.show()
    K1=corr.index(max(corr))*2+1#记录准确率最高的K
    return K1

#print(decide_label_Euclidean(data_process.train_data,data_process.val_
data))
#print(decide_label_Chebyshev(data_process.train_data,data_process.val_
data))
#print(decide_label_Manhattan(data_process.train_data,data_process.val_
data))


def Euclidean_classify(train,test):
    '''用Euclidean_distance对测试集分类'''
    total=data_process.count_total(test)
    K=1
    a=np.array(test)
    t_data=a[:,0:4]
    predict = [] # 对测试集各样本进行分类
    for sample in t_data:
        sort = Euclidean_distance(train, sample)
        lab_0 = 0
        lab_1 = 0
        for i in range(K):
            if sort[i][1] == 0:
                lab_0 += 1
            else:
                lab_1 += 1
        if lab_0 > lab_1:
            predict.append(0)
        else:
            predict.append(1)
    Euclidean_result=list(test)
    for i in range(total):
        Euclidean_result[i].append(predict[i])
    title=['Recency (months)','Frequency (times)','Monetary (c.c. blood
)','Time (months)','My prediction']
    Euclidean_result.insert(0,title)
    submit = pd.DataFrame(data=Euclidean_result)
    #print(submit)
    submit.to_csv('./task1_test_Euclidean.csv',encoding='gbk', header=N
one, index=None)
Euclidean_classify(data_process.train_data,data_process.test_data)
```

```
def Chebyshev_classify(train,test1):
    '''用 Chebyshev_distance 对测试集分类'''
    total=data_process.count_total(test1)

    K=1
    a=np.array(test1)
    t_data=a[:,0:4]
    predict = [] # 对测试集各样本进行分类
    for sample in t_data:
        sort = Chebyshev_distance(train, sample)
        lab_0 = 0
        lab_1 = 0
        for i in range(K):
            if sort[i][1] == 0:
                lab_0 += 1
            else:
                lab_1 += 1
        if lab_0 > lab_1:
            predict.append(0)
        else:
            predict.append(1)
    Chebyshev_result=list(test1)
    for i in range(total):
        Chebyshev_result[i][4]=predict[i]
    title=['Recency (months)','Frequency (times)','Monetary (c.c. blood
)','Time (months)','My prediction']
    Chebyshev_result.insert(0, title)
    submit1 = pd.DataFrame(data=Chebyshev_result)
    #print(submit)
    submit1.to_csv('./task1_test_Chebyshev.csv',encoding='gbk', header=
None, index=None)
Chebyshev_classify(data_process.train_data,data_process.test_data)

def Manhattan_classify(train,test2):
    '''用 Manhattan 对测试集分类'''
    total=data_process.count_total(test2)
    K=1
    a=np.array(test2)
    t_data=a[:,0:4]
    predict = [] # 对测试集各样本进行分类
    for sample in t_data:
        sort = Manhattan_distance(train, sample)
        lab_0 = 0
```

```
        lab_1 = 0
        for i in range(K):
            if sort[i][1] == 0:
                lab_0 += 1
            else:
                lab_1 += 1
        if lab_0 > lab_1:
            predict.append(0)
        else:
            predict.append(1)
    Manhattan_result=list(test2)
    for i in range(total):
        Manhattan_result[i][4]=predict[i]
    title=['Recency (months)','Frequency (times)','Monetary (c.c. blood
)','Time (months)','My prediction']
    Manhattan_result.insert(0, title)
    submit2 = pd.DataFrame(data=Manhattan_result)
    #print(submit)
    submit2.to_csv('./task1_test_Manhattan.csv',encoding='gbk', header=
None, index=None)
Manhattan_classify(data_process.train_data,data_process.test_data)
```

task_2.py

```
#coding :utf-8

'''任务二：利用马氏距离作为 KNN 算法的度量函数，对测试集进行分类。'''

import numpy as np
import pandas as pd
import math
import matplotlib.pyplot as plt
np.seterr(invalid='ignore')

import data_process

#利用梯度更新方法更新马氏距离的可学习参数 A
class UPA():
    def __init__(self, learning_rate=0.001, epoch=500):
        self.learning_rate = learning_rate
        self.epoch = epoch

    def train(self, X, Y):
        (n, d) = X.shape
```

```
        self.n_samples = n  # 样本数
        self.dim = d  # 样本点的维度

        # A:d*e
        e = 2  # A 的维度，在距离度量时实际进行了降维
        A_shape = (d, e)
        # 初始化特征矩阵 A
        self.A = 0.1*np.random.standard_normal(size = A_shape)
        # 利用梯度下降训练参数 A
        count = 0  # 迭代的计数器
        f = 0  # 优化目标 f(A)=sum_i(p_i)
        f_list = []
        while count < self.epoch:
            if count % 5 == 0 and count > 1:
                print('Step{},f(a)={}...'.format(count, f))
            # 计算 AX:e
            XA = np.dot(X, self.A)
            # 距离度量
            sum_row = np.sum(XA ** 2, axis=1)
            XAATXT = np.dot(XA, XA.transpose())
            dist_mat = sum_row + np.reshape(sum_row, (-
1, 1)) - 2 * XAATXT
            # 将距离转变为概率
            exp_neg_dist = np.exp(-dist_mat)
            exp_neg_dist = exp_neg_dist - np.diag(np.diag(exp_neg_dist)
)
            prob_mat = exp_neg_dist / np.sum(exp_neg_dist, axis=1).resh
ape((-1, 1))
            # pi = sum_{j  in C_i}p_{ij} pi 构成的矩阵
            prob_row = np.array([np.sum(prob_mat[i][Y == Y[i]]) for i i
n range(self.n_samples)])
            f = np.sum(prob_row)
            if np.isnan(f):
                break
            f_list.append(f)
            # A 的梯度，即 f 对 A 的偏导
            gradients = np.zeros((self.dim, self.dim))
            # 对 i 的循环作为外层循环
            for i in range(self.n_samples):
                # 梯度中的第一项
                first_item = np.zeros((self.dim, self.dim))
                # 梯度中的第二项
                second_item = np.zeros((self.dim, self.dim))
                # 对 j 的循环作为内层循环
```

```
                for k in range(self.n_samples):
                    out_prob = np.outer(X[i] - X[k], X[i] - X[k])
                    first_item += prob_mat[i][k] * out_prob
                    if Y[k] == Y[i]:
                        second_item += prob_mat[i][k] * out_prob
                gradients += prob_row[i] * first_item - second_item
            gradients = 2 * np.dot(gradients, self.A)
            # 利用梯度更新A
            self.A += self.learning_rate * gradients
            # 循环计数器次数+1
            count += 1
        return self.A
NCA_model = UPA()
a=np.array(data_process.train_data)
X=a[:,0:4]
Y=a[:,4]
A=NCA_model.train(X,Y)
#print(A)

def Mahalanobis_distance(train_data,data):
    '''计算单个样本与训练集各个样本间的马氏距离，并按升序排列'''
    A1=np.array(A)
    a1 = np.array(train_data)
    a2 = np.array(data)
    td = a1[:, 0:4]
    d = a2[0:4]
    label = a1[:, 4]
    C_distance = []
    dis_lab = []  # 存储该样本与训练样本的马氏距离和训练样本对应的类别
    for sample in td:
        temp=np.dot(d-sample,A1)
        temp_tr=np.transpose(temp)
        mul=np.dot(temp,temp_tr)
        norm = math.sqrt(mul)
        C_distance.append(norm)
    for i in range(data_process.count_total(train_data)):
        dis_lab.append([C_distance[i], label[i]])
    sort = sorted(dis_lab, key=(lambda x: x[0]))  # 将马氏距离按升序排列
    return sort


def decide_label(train_data,val_data):
    '''用验证集计算各个K下的正确率'''
    total=data_process.count_total(val_data)
    a=np.array(val_data)
```

```
    data=a[:,0:4]
    label=a[:,4]
    corr = []#各个K 下的正确率
    for K in range(1,30,2):
        predict = []  # 对验证集各样本进行分类
        for sample in data:
            sort = Mahalanobis_distance(train_data, sample)
            lab_0=0
            lab_1=0
            for i in range(K):
                if sort[i][1]==0:
                    lab_0+=1
                else:
                    lab_1+=1
            if lab_0>lab_1:
                predict.append(0)
            else:
                predict.append(1)
        correct=0
        for i , lab in enumerate(label):
            if predict[i]==lab:
                correct+=1
        corr.append(correct/total)
    #绘制K 对精度影响的曲线图
    K=list(range(1,30,2))
    fig=plt.figure()
    ax = fig.add_subplot(1, 1, 1)
    fig.suptitle('K of Mahalanobis', fontsize = 14, fontweight='bold')
    ax.set_xlabel("K")
    ax.set_ylabel("correction rate")
    plt.plot(K, corr)
    plt.show()
    K2=corr.index(max(corr))*2+1#记录准确率最高的K
    return K2
#print(decide_label(data_process.train_data,data_process.val_data))

def Mahalanobis_classify(train,test):
    '''用Mahalanobis 对测试集分类'''
    total=data_process.count_total(test)
    #K=Manhattan.decide_label(train,val_data)
    K=1
    a=np.array(test)
    t_data=a[:,0:4]
    predict = [] # 对测试集各样本进行分类
```

```
    for sample in t_data:
        sort = Mahalanobis_distance(train, sample)
        lab_0 = 0
        lab_1 = 0
        for i in range(K):
            if sort[i][1] == 0:
                lab_0 += 1
            else:
                lab_1 += 1
        if lab_0 > lab_1:
            predict.append(0)
        else:
            predict.append(1)
    Mahalanobis_result=list(test)
    for i in range(total):
        Mahalanobis_result[i].append(predict[i])
        #Mahalanobis_result[i][4]=predict[i]
    title=['Recency (months)','Frequency (times)','Monetary (c.c. blood
)','Time (months)','My prediction']
    Mahalanobis_result.insert(0, title)
    submit2 = pd.DataFrame(data=Mahalanobis_result)
    #print(submit)
    submit2.to_csv('./task2_test_prediction.csv', encoding='gbk', heade
r=None, index=None)

Mahalanobis_classify(data_process.train_data,data_process.test_data)
```

心得体会

本学习的模式识别课程实验我选择的三个实验分别是：贝叶斯分类任务、隐马尔可夫分词模型任务和 KNN 分类任务。

贝叶斯分类任务以模式识别课程最初所学的贝叶斯公式为基础拓展，结合了概率论所学的高斯分布知识，设计出朴素贝叶斯分类器并对具有 13 个维度特征的 wine 数据集进行分类。隐马尔可夫分词任务则是对课程中所学的隐马尔可夫模型的应用，为日后学习自然语言处理奠定了基础。KNN 分类任务则是对课堂中所学非参数技术中涉及的四种距离度量方法的应用以及梯度下降法的实践。

通过完成三个实验，我对于课堂上所学的理论知识在实际数据集以及场景上的应用有了更加深刻的体会。同时，我的 python 编程能力也在实验中得到了提升。