模式识别实验报告

专业：	人工智能
学号：	61518407
年级：	2018
姓名：	李浩瑞

签名：

时间：

实验一 隐马尔可夫分词

1. 问题描述

利用隐马尔科夫模型进行中文语句的分词。

任务：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：

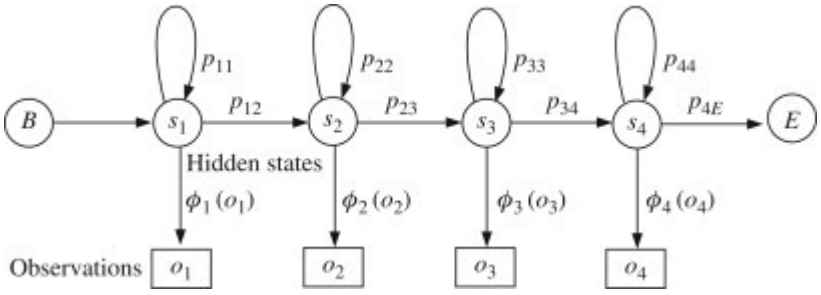
1) 今天的天气很好。

2) 学习模式识别课程是有难度的事情。

3) 我是东南大学的学生。

2. 实现步骤与流程

隐马尔可夫模型（HMM）描述了含有隐藏变量的马尔可夫随机过 程，该模型涉及两个序列和三个概率矩阵，即可观察的观测序列 V^{T} 、隐藏的状态序列 ω^{T} 、初始的状态概率矩阵 π, 状态转移概率矩阵 A 及状态生成观测的概率矩阵 B。HMM 可表示为 $\{\mathrm{V}^{\mathrm{T}}, \omega^{\mathrm{T}}, \pi, \mathrm{A}, \mathrm{B}, \lambda=(\pi, \mathrm{A}, \mathrm{B})\}, \lambda=(\pi, \mathrm{A}, \mathrm{B})$ 决定 HMM 模型， π 和 A 决定观测序列， B 决定状态序列。



图表 1：HMM 模型示意图

HMM 具有三个基本问题：概率计算问题、学习问题和预测

问题。概率计算问题是计算在模型 λ 下观测序列 V^{T} 的概率，直接求解的方法不可行，计算量非常大，有效的方法是前向一后向算法。学习问题是已知观测V^{T} 估计模型 $\lambda=(\pi, \mathrm{A}, \mathrm{B})$ 的参数，预测问题是给定观测序列，求出最有可能的对应的状态序列，可用近似算法和 Viterbi 算法。

训练时，通过统计语料库中相关信息训练 HMM 中的三个参数 PI、A 和 B。A 表示字的词位状态转移矩阵，B 表示词位到词的混淆矩阵。从语料库中可以获得每个词位出现的次数，每个字符出现的次数。

通过频率代替概率得到三个参数的值：

$$A_{ij}=P\left(Z_j / Z_i\right)=\frac{P\left(Z_i, Z_j\right)}{P\left(Z_i\right)} \approx \frac{\operatorname{freq}\left(Z_i, Z_j\right)}{\operatorname{freq}\left(Z_i\right)}$$

$$B_{ij}=P\left(O_j / Z_i\right)=\frac{P\left(O_j, Z_i\right)}{P\left(Z_i\right)} \approx \frac{\operatorname{freq}\left(O_j, Z_i\right)}{\operatorname{freq}\left(Z_i\right)}$$

转移概率矩阵为：

$$A=\begin{bmatrix} 0.0 & p(M/B) & p(E/B) & 0.0 \\ 0.0 & p(M/M) & p(E/M) & 0.0 \\ p(B/E) & 0.0 & 0.0 & p(S/E) \\ p(B/S) & 0.0 & 0.0 & p(S/S) \end{bmatrix}$$

预测时，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。

$$\delta_{t+1}=\max_{1 \leqslant j \leqslant N}\left[\delta_t(j) a_{ij}\right] b_i\left(o_{t+1}\right)$$

$$\psi_t = \arg \max_{1 \leqslant j \leqslant N} \left[\delta_{t-1}(j) a_{ji}\right], i = 1,2,\cdots,N$$

3. 实验结果与分析

针对实验要求中的数据和个人额外增加的测试数据：

```
在第三世界读书的我废寝忘食穷困潦倒不吃不喝三个月写出了可以正确进行隐马尔可夫分词的人工智能程序！
烧花鸭烧雏鸡烧子鹅烧鹿尾巴儿
```

分词结果如下：

```
mytestoutput: 今天 的 天气 很 好 。
学习 模式 识别 课程 是 有 难度 的 事情 。
我是 东南 大学 的 学生 。
在 第三 世界 读书 的 我 废寝忘食 穷 困潦 倒 不 吃 不 喝 三 个月 写出 了 可以 正确 进行 隐 马尔 可夫 分词 的 人工 智能 程序 ！
烧 花 鸭 烧 雏 鸡 烧子 鹅 烧 鹿 尾 巴儿
```

经过多轮测试，发现程序对大部分语句的分词效果非常棒，但许多四字词语会被分成两个词，即使是在原本的“人民日报”数据集中出现过的四字词语：“*穷困潦倒*”、“*第三世界*”等词也无法正确分类。而一些常见文本由于特殊使用环境也会分词错误，如“*尾巴儿*”、“*烧雏鸡*”等口语、艺术作品词。

个人思考觉得，可能的原因有两点：

在于 HMM 算法本身，由于概率本身不断传递，连乘结果不断变小，因此连续的状态很难一直保持。这就导致出现了大量的二字词云，甚至孤单的一个字。

另一个原因在于人民日报文本数量小，且有报纸期刊文章的特殊性，数据更偏向社会民生，对于口语和艺术作品中会使用的词，如“*尾巴儿*”、“*烧花鸭*”，无法覆盖。如果有更多元的数据来源，相信训练效果会更好。

4. 代码附录

```
"""
使用已分词的语料进行训练
使用维特比算法对测试集进行分词
输出分词结果并写入 data 文件夹下的 output.txt

61518407 李浩瑞 2021.1.1
"""

import pandas as pd
from numpy import *
from math import log
import os
import sys
import json

"""设置全局变量"""
# 读写文件索引
project_path = os.path.dirname(os.path.abspath(__file__))   # 获取当前文件路径的上一级目录
train_path = project_path+r'\data\RenMinData.txt_utf8'  # 拼接训练路径字符串
test_path = project_path+r'\data\mytest.txt'    # 拼接测试路径字符串
output_path = project_path+r'\output.txt'
# 全局变量
States_index = [0, 1, 2, 3]  # 状态索引
STATES = ['B', 'M', 'E', 'S']  # 单个汉字的四种状态
A = {}
B = {}
Pi = {}
word_set = set()  # 所有显式状态的集合
count_dic = {}
neg_infinity = -61518407e+100  # 负无穷表示 log(0)
line_num = 0


def Tag(word):
    """
    获取训练集中每段词语的状态标签
    word:list
```

```
    tag: list
    """
    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


def train(trainset, word_set, Pi, A, B):
    """
    void
    根据训练数据 trainset 统计频数，
    将所有训练集内出现过的词存入 word_set，
    将概率存入 Pi,A,B
    """
    # 初始化字典
    for i in STATES:
        Pi[i] = 0.0
        A[i] = {}
        B[i] = {}
        count_dic[i] = 0
        for j in STATES:
            A[i][j] = 0.0

    # 统计频数
    for line in trainset:
        '''读取训练集'''
        line = line.strip()  # 不 strip 会读到\n
        global line_num
        line_num += 1
        word_list = []
        for k in range(len(line)):
            if line[k] == ' ':
                continue
            word_list.append(line[k])
        word_set = word_set | set(word_list)  # “集合或”，获得训练集所有字的集合
        '''从训练集计算状态转移矩阵'''
```

```
            line = line.split(' ')
            line_state = []
            for i in line:
                line_state.extend(Tag(i))

            Pi[line_state[0]] += 1
            for j in range(len(line_state)-1):
                A[line_state[j]][line_state[j+1]] += 1
            for p in range(len(line_state)):
                count_dic[line_state[p]] += 1
                for state in STATES:
                    if word_list[p] not in B[state]:
                        B[state][word_list[p]] = 0.0
                B[line_state[p]][word_list[p]] += 1
    #log(概率)
    #对概率 0 取无穷小 neg_infinity
    for i in STATES:
        if Pi[i] == 0:
            Pi[i] = neg_infinity
        else:
            Pi[i] = math.log(Pi[i] / line_num)
        for j in STATES:
            if A[i][j] == 0:
                A[i][j] = neg_infinity
            else:
                A[i][j] = math.log(A[i][j] / count_dic[i])
        for j in B[i]:
            if B[i][j] == 0:
                B[i][j] = neg_infinity
            else:
                B[i][j] = math.log(B[i][j] / count_dic[i])


def Viterbi(obs, Pi, A, B):
    """
    return path[state]:list 存储句子的隐状态
    已知 Pi,A,B,显状态 sentence
    求测试集最有可能的隐状态集合
    注意概率已经取过 log,乘变加
    """
    V = [{}]  # 动态规划表 tabular
    path = {}
    if obs[0] not in B['B']:
        # 防止没见过的隐状态，设置为 S，不可能 B,M,E,减少传递过程中计算错误的
可能
```

```
        for i in STATES:
            if i == 'S':
                B[i][obs[0]] = 0
            else:
                B[i][obs[0]] = neg_infinity
    for i in STATES:
        V[0][i] = Pi[i] + B[i][obs[0]]
        path[i] = [i]
    for i in range(1, len(obs)):
        V.append({})
        new_path = {}

        for state0 in STATES:
            items = []
            for state1 in STATES:
                if obs[i] not in B[state0]:
                    if obs[i-1] not in B[state0]:
                        prob = V[i - 1][state1] + A[state1][state0]
                    else:
                        prob = V[i - 1][state1] + A[state1][state0]
                else:
                    prob = V[i-1][state1] + A[state1][state0] + \
                        B[state0][obs[i]]
                items.append((prob, state1))
            best = max(items)
            V[i][state0] = best[0]
            new_path[state0] = path[best[1]] + [state0]
        path = new_path
    prob, state = max([(V[len(obs) - 1][state], state) for state in STA
TES])
    return path[state]


def test(testset, Pi, A, B):
    """
    void
    对测试数据进行分词
    输出分词结果并写入 output.txt
    testset = open(test_path)
    Pi,A,B:doc
    output:list
    """
    output = ''
    for line in testset:
        line = line.strip()
```

```
        tag = Viterbi(line, Pi, A, B)
        splited = []
        start = -1
        started = False
        if len(tag) != len(line):
            return None
        elif len(tag) == 1:
            splited.append(line[0])  # 语句只有一个字，直接输出
        else:
            for i in range(len(tag)):
                if tag[i] == 'S':
                    if started:
                        started = False
                        splited.append(line[start:i])
                    splited.append(line[i])
                elif tag[i] == 'B':
                    if started:
                        splited.append(line[start:i])
                    start = i
                    started = True
                elif tag[i] == 'E':
                    started = False
                    word = line[start:i + 1]
                    splited.append(word)
                elif tag[i] == 'M':
                    continue
        list = ''
        for i in range(len(splited)):
            list = list + splited[i] + ' '
        output = output + list + '\n'
    print("mytestoutput:", output)
    outputfile = open(output_path, mode='w', encoding='utf-8')
    outputfile.write(output)


if __name__ == '__main__':
    """进行训练"""
    trainset = open(train_path, encoding='utf-8')  # 读取训练集
    train(trainset, word_set, Pi, A, B)
    """保存模型"""
    with open(project_path+r"model_Pi.json", "w") as f:
        f.write(json.dumps(Pi, ensure_ascii=True, indent=4, separators=
(',', ':')))
    with open(project_path+r"model_A.json", "w") as f:
```

```
        f.write(json.dumps(A, ensure_ascii=True, indent=4, separators=(
',', ':')))
    with open(project_path+r"model_B.json", "w") as f:
        f.write(json.dumps(B, ensure_ascii=True, indent=4, separators=(
',', ':')))
    """进行测试"""
    testset = open(test_path, encoding='utf-8')  # 读取测试集
    test(testset, Pi, A, B)
```

实验二 线性分类器

1. 问题描述

利用线性分类器对 MNIST 数据集中的测试集进行分类。

任务一：对 MNIST 数据集做恰当预处理，在处理后的训练集上学习一个多类线性分类器，并对处理后的测试集进行分类。

任务二：利用 LDA 降维技术进行对 MNIST 数据集的降维工作，在降维后的数据集上完成多类线性分类器的训练和测试。要求比较应用 LDA 降维技术前后，分类器准确率的变化。（对于降维后的数据，可以尝试利用可视化方法展示结果。）

2. 实现步骤与流程

2.1 数据预处理和增广

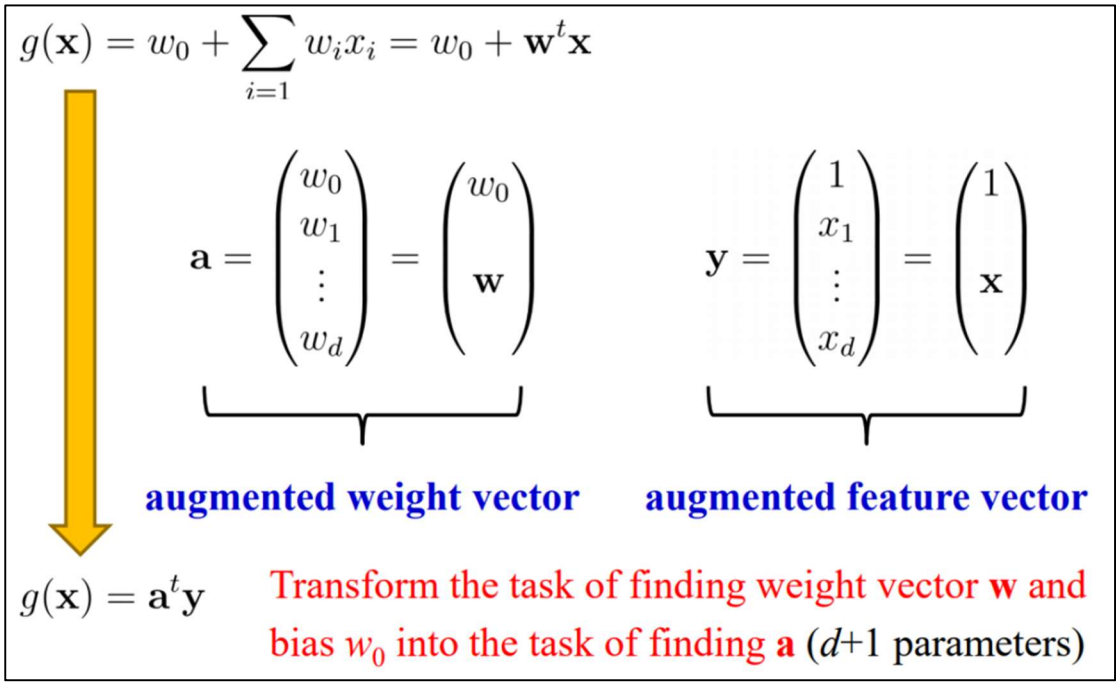
原始数据为.ubyte 数据，无法直接读取，需要进行解码。

同时由于提供的 Label 是 int 型，需要转换为 one-hot 编码。

对与线性判别器：

$$f(x_i, W, b) = Wx_i + b$$

为方便训练，本实验对数据和权重矩阵做增广处理，将问题修改为训练 (d+1, N) 维权重矩阵，其中 d 为特征维度，N 为类别数量:



图表 2:增广处理

因此最终输入线性分类器的是(60000,28*28+1)大小的训练数据。

2.2 设计损失函数与更新

考虑到本次实验无法使用 Pytorch，TensorFlow 等高级工具包，因此损失函数需要尽量好求导，以便手工编写。

选取的损失函数为 MSE Loss：

$$\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

对 y_hat 求导结果为简单地将二次拿下来，减少编程难度。

获得导数之后，采用梯度下降的方法更新曾广后的权值。

同时，为了便于找到最优解，本实验还设计了变学习率的更新方法，如果损失函数上升巨大，或者长时间不下降，则自动调小学习率。

经过 150 轮训练，正确率达到了 90.6%。

由于训练时间太长，因此在训练结束后设计了保存权重 w 为.npy 文件，下次可以直接读取使用，跳过训练时间。

2.3 LDA 算法

LDA 是一种有监督的降维，的目标是要将数据在低维度上进行投影，投影后希望每一种类别数据的投影点尽可能的接近，而不同类别的数据的类别中心之间的距离尽可能的大。

本任务中我们需要多类向低维投影,。假设我们投影到的低维空间的维度为 d, 对应的基向量为 $(\mathrm{w}_1, \mathrm{w}_2, \ldots \mathrm{w}_\mathrm{d})$,基向量组成的矩阵为 W 它是一个 $\mathrm{n} \times \mathrm{d}$的矩阵。优化目标为：

$$\frac{W^T S_b W}{W^T S_w W}$$

其中 S 为散布矩阵：

$$S_b = \sum_{j=1}^{k} N_j (\mu_j - \mu)(\mu_j - \mu)^T$$

$$S_w = \sum_{j=1}^{k} \sum_{x \in X_j} (x - \mu_j)(x - \mu_j)^T$$

计算 $\mathrm{S_w^{-1}S_b}$ 的最大的 d 个特征值和对应的 d 个特征向量就得到投影矩阵 W，可以将样本转换为新的样本。

在实际测试中，发现因为 MNIST 数据中存在大量的 0，会导致矩阵不满秩无法求逆，因此使用了伪逆。

在计算出投影矩阵后，对 x 进行降维，重新训练。150 轮后正确率为 87.8%。

3. 实验结果与分析

普通算法最终收敛于 90.6%的正确率（150 轮），LDA 后的数据收敛于 87.8%。

“实验要求”中鼓励进行数据可视化，因此从测试集中随机提取一些数字进行判断：

图表 3：可视化和验证准确率

可以看到分类结果正确无误。

4. 代码附录

```
"""
使用线性分类器分类 MNIST
LDA 降维

61518407 李浩瑞 1.10
"""

import gzip
import pickle
import os,sys
import struct
import numpy as np
def decode_idx3_ubyte(idx3_ubyte_file):
    """
    解析 idx3 文件的通用函数
    :param idx3_ubyte_file: idx3 文件路径
    :return: 数据集
    """
    # 读取二进制数据
    bin_data = open(idx3_ubyte_file, 'rb').read()
    # 解析文件头信息，依次为魔数、图片数量、每张图片高、每张图片宽
    offset = 0
    fmt_header = '>iiii' #因为数据结构中前 4 行的数据类型都是 32 位整型，所以采用 i 格式，但我们需要读取前 4 行数据，所以需要 4 个 i。我们后面会看到标签集中，只使用 2 个 ii。
    magic_number, num_images, num_rows, num_cols = struct.unpack_from(fmt_header, bin_data, offset)
    print('图片数量: %d 张, 图片大小: %d*%d' % (num_images, num_rows, num_cols))
    # 解析数据集
    image_size = num_rows * num_cols
    offset += struct.calcsize(fmt_header)  #获得数据在缓存中的指针位置，从前面介绍的数据结构可以看出，读取了前 4 行之后，指针位置（即偏移位置 offset）指向 0016。
    print(offset)
    fmt_image = '>' + str(image_size) + 'B'  #图像数据像素值的类型为 unsigned char 型，对应的 format 格式为 B。这里还有加上图像大小 784，是为了读取 784 个 B 格式数据，如果没有则只会读取一个值（即一副图像中的一个像素值）
    print(fmt_image,offset,struct.calcsize(fmt_image))
    images = np.empty((num_images, num_rows, num_cols))
    #plt.figure()
    for i in range(num_images):
        if (i + 1) % 10000 == 0:
            print('已解析 %d' % (i + 1) + '张')
```

```
            print(offset)
        images[i] = np.array(struct.unpack_from(fmt_image, bin_data, of
fset)).reshape((num_rows, num_cols))
        #print(images[i])
        offset += struct.calcsize(fmt_image)
    return images

def decode_idx1_ubyte(idx1_ubyte_file):
    """
    解析 idx1 文件的通用函数
    :param idx1_ubyte_file: idx1 文件路径
    :return: 数据集
    """
    # 读取二进制数据
    bin_data = open(idx1_ubyte_file, 'rb').read()
    # 解析文件头信息，依次为魔数和标签数
    offset = 0
    fmt_header = '>ii'
    magic_number, num_images = struct.unpack_from(fmt_header, bin_data,
 offset)
    print('图片数量: %d 张' % ( num_images))
    # 解析数据集
    offset += struct.calcsize(fmt_header)
    fmt_image = '>B'
    labels = np.empty(num_images)
    for i in range(num_images):
        if (i + 1) % 10000 == 0:
            print ('已解析 %d' % (i + 1) + '张')
        labels[i] = struct.unpack_from(fmt_image, bin_data, offset)[0]
        offset += struct.calcsize(fmt_image)
    return labels


#读写文件索引
project_path = os.getcwd()   # 获取当前文件路径的上一级目录
train_image_path = project_path+r'\MNIST_data\train-images.idx3-
ubyte'  # 拼接训练路径字符串
train_label_path = project_path+r'\MNIST_data\train-labels.idx1-
ubyte'    # 拼接训练路径字符串
test_image_path = project_path+r'\MNIST_data\t10k-images.idx3-
ubyte'  # 拼接训练路径字符串

#读取数据
image = decode_idx3_ubyte(train_image_path)
label = decode_idx1_ubyte(train_label_path)
```

```
test_image=decode_idx3_ubyte(test_image_path)

train_X=[]
for item in image:
    temp=item.reshape(1,28*28)
    temp=np.append(temp,1)
    train_X.append(temp)
train_Y=[]
for item in label:
    temp=np.array([0 for i in range(10)])
    temp[int(item)]=1
    train_Y.append(temp)

train_x=np.array(train_X)
train_y=np.array(train_Y)
epoch=100
D_in = 785
D_out = 10
learning_rate=5e-7
loss=0
w1 = np.random.rand(D_in,D_out)
b = np.random.rand(D_out)
x=train_x
y=train_y

def softmax_pred(y_pred):
    y_soft_pred=[]
    for item in y_pred:
        item-=np.max(item)
        y_soft_pred_item=(np.exp(item) / np.sum(np.exp(item)))
        #print(y_soft_pred_item)
        y_soft_pred.append(y_soft_pred_item)
    return y_soft_pred

def val(y_soft_pred,y):
    r,w=0,0
    for i in range(0,60000):
        item_pred,item_real = y_soft_pred[i],y[i]
        list1 = item_pred.tolist()
        max_index1 = list1.index(max(list1))+1
        list2 = item_real.tolist()
        max_index2 = list2.index(max(list2))+1
        if max_index1 == max_index2:
            r+=1
```

```
        else:
            w+=1
    return (r/(r+w))

for it in range(epoch):
    #forward pass 前向
    y_pred=x.dot(w1)
    e=(y_pred-np.max(y_pred))
    y_pred=x.dot(w1)
    y_soft_pred=softmax_pred(y_pred)
    #定义 loss
    #MSE loss，均方误差
    #MSE 是非凸的，所以不一定会找到全局最优，梯度下降过程中也可能先下后上再下
    pre_loss=loss
    loss=np.square(y_soft_pred - y).sum()
    if ((it % 2)==0):
        print("progress:{0}%".format(round((it + 1) * 100 / epoch)),"it
er:",it,"Loss:",loss)
        if (np.square(pre_loss-loss)<100 or (pre_loss-loss)<-5000):
            learning_rate*=0.9
            print("lr change")
    grad_Loss_yhat=2.0*(y_soft_pred - y)
    grad_Loss_w1=x.T.dot(grad_Loss_yhat)
    # Update weights
    w1 -= learning_rate * grad_Loss_w1
    if ((it % 20)==0):
        print("grad_Loss_w1:",grad_Loss_w1)
        acc=val(y_soft_pred,y)
        print("now acc is:",acc)

finalacc=val(y_soft_pred,y)
np.save(project_path+r"my_acc{}_w.npy".format(str(finalacc)[:5]),w1)
print("Saved:",project_path+r"my_acc{}_w.npy".format(str(finalacc)[:5])
)

"""LDA"""
def LDA(X,y):
    n = 60000
    Dim = 28*28
    dim = 9
    N = 10
    m = np.mean(X,axis=0).reshape(Dim,1)
    X = [[]for i in range((10))]
    mean = []
```

```
    S_B,S_W = np.zeros((Dim,Dim)),np.zeros((Dim,Dim))
    for i in range(n):
        X[y[i]].append(X[i])
        print("X[i]",X[i].shape)
        print("first i",i/n)
    for i in range(N):
        X[i] = np.array(X[i])
        mean.append(np.mean(X[i], axis=0).reshape(Dim, 1))
        print("second i",i/N)
    for i in range(N):
        S_B += np.matmul((mean[i]-m).reshape(len(m),1)) , ((mean[i]-
m).reshape(len(m),1).T)
        print("third i",i/N)
    for i in range(N):
        print("forth i",i)
        for j in range(len(X[i])):
            S_W +=np.matmul(((X[i][j].reshape(len(X[i][j]),1)-
mean[i])),((X[i][j].reshape(len(X[i][j]),1)-mean[i]).T))
    try:
        S = np.matmul((np.linalg.inv(S_W)) , S_B)
    except:
        S = np.matmul(np.linalg.pinv(S_W) , S_B)

    eigVal, eigVec = np.linalg.eig(S)

    idx = eigVal.argsort()[::-1]
    eigVal = eigVal[idx]
    eigVec = eigVec[:,idx]

    va = np.zeros((dim,X.shape[1]))
    for i in range(dim):
        va[i] = eigVec[:, i]

    return va

train_X=[]
for item in image:
    temp=item.reshape(28*28,1)
    train_X.append(temp)
train_LDA_X=[]
for item in va:
    print(item)
    temp=item.reshape(1,28*28)
    temp=np.append(temp,1)
```

```
    train_LDA_X.append(temp)

train_X=np.array(train_X)
train_y = np.array([int(y) for y in label])
va = LDA(train_X,train_y)

epoch=150
D_in = 10
D_out = 10
learning_rate=5e-7
loss=0
w2 = np.random.rand(D_in,D_out)
train_falt_X=np.array(train_falt_X)
y=train_y
for it in range(epoch):
    y_pred=train_falt_X.dot(w2)
    e=(y_pred-np.max(y_pred))
    y_soft_pred=softmax_pred(y_pred)
    pre_loss=loss
    loss=np.square(y_soft_pred - y).sum()#都是 N*10
    if ((it % 2)==0):
        print("progress:{0}%".format(round((it + 1) * 100 / epoch)),"it
er:",it,"Loss:",loss)#,end="\r"
        if (np.square(pre_loss-loss)<100 or (pre_loss-loss)<-5000):
            learning_rate*=0.9
            print("lr change")
    grad_Loss_yhat=2.0*(y_soft_pred - y)
    grad_Loss_w2=train_falt_X.T.dot(grad_Loss_yhat)
    w2 -= learning_rate * grad_Loss_w2
    if ((it % 20)==0 or it == 149):
        print("grad_Loss_w1:",grad_Loss_w1)
        acc=val(y_soft_pred,y)
        print("now acc is:",acc)
finalacc=val(y_soft_pred,y)
np.save(project_path+r"my_LDA_acc{}_w.npy".format(str(finalacc)[:5]),w1
)
print("Saved:",project_path+r"my_LDA_acc{}_w.npy".format(str(finalacc)[
:5]))
```

实验三 KNN

5. 问题描述

任务一：利用欧式距离作为 KNN 算法的度量函数，对测试集进行分类。实验报告中，要求在验证集上分析近邻数 k 对 KNN 算法分类精度的影响。

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，实验报告中请对优化过程的梯度计算公式进行推导，即给出$\frac{\partial f}{\partial A}$

6. 实现步骤与流程

2.1 欧式距离 KNN 分类

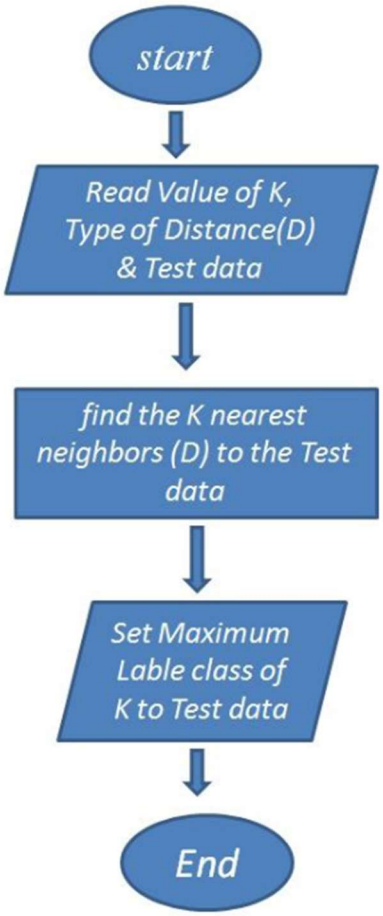
(1)算法思想简述

在多维空间中，欧氏距离是：

$$d(x,y):=\sqrt{(x_1-y_1)^2+(x_2-y_2)^2+\cdots+(x_n-y_n)^2}$$

$$=\sqrt{\sum_{i=1}^{n}(x_i-y_i)^2}$$

对于本次实验，在训练集中数据和标签已知的情况下，输入测试数据，将测试数据的特征与训练集中对应的特征进行相互比较，找到训练集中与之最为相似的前 K 个数据，则该测试数据对应的类别就是 K 个数据中出现次数最多的那个分类，流程图为：



图表 4：KNN 分类流程

（2）预处理

wine 葡萄酒数据集是 UCI 上的公开数据集。数据集包含由三种不同葡萄酿造 的葡萄酒，通过化学分析确定了葡萄酒中含有的 13 种成分的含量。数据集的相关信息如下图：

Data Set Characteristics:	Multivariate	**Number of Instances:**	178	**Area:**	Physical
Attribute Characteristics:	Integer, Real	**Number of Attributes:**	13	**Date Donated**	1991-07-01
Associated Tasks:	Classification	**Missing Values?**	No	**Number of Web Hits:**	1578807

图表 5：葡萄酒数据集属性

数据集已被划分为训练集、验证集和测试集，分别存储于 data 文件夹中的 train_data.mat, val_data.mat, test_data.mat

发现在训练集中最后一维的特征存在缺失，且数据彼此之间相差大，因此我们需要将数据进行补齐或删除，并标准化。

本次实验中使用了数据补齐+标准化的方法进行预处理。

2.2 马氏距离实现过程

（1）马氏距离优化过程推导

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：

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

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$\boldsymbol{M}$的半正定性质，可将上述定义表述为：

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

$$= \left|\left| \mathrm{Ax_i} - \mathrm{Ax_j} \right|\right|_2$$

可以理解为先把数据降维到低维空间，然后再求欧氏距离。

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：

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

其中，C_i表示与样例$\boldsymbol{x}_i$同类的样例集合，p_{ij} 定义为：

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

方便起见，我们记$g_{ij} = exp\left(-|Ax_i - Ax_j|^2\right)$，那么有

$$p_{ij} = \frac{g_{ij}}{\sum_{k \neq i} g_{ik}}$$

求导：

$$\frac{\partial \mathrm{g_{ij}}}{\partial \mathrm{A}} = -2g_{ij}A(x_i - x_j)(x_i - x_j)^T$$

同理：

$$\frac{\partial \mathrm{p_{ij}}}{\partial \mathrm{A}} = \frac{1}{(\sum_{k\neq i} g_{ik})^2}\left(\frac{\partial \mathrm{g_{ij}}}{\partial \mathrm{A}}\sum_{k\neq i} g_{ik} - g_{ij}\sum_{k\neq i}\frac{\partial \mathrm{g_{ij}}}{\partial \mathrm{A}}\right)$$

$$\frac{1}{(\sum_{k\neq i} g_i k)^2}\frac{\partial \mathrm{g_{ij}}}{\partial \mathrm{A}}\sum_{k\neq i} g_{ik} = \frac{-2g_{ij}A(x_i - x_j)(x_i - x_j)^T}{\sum_{k\neq i} g_{ik}} = -2p_{ij}A(x_i - x_j)(x_i - x_j)^T$$

$$\frac{g_{ij}}{(\sum_{k\neq i} g_{ik})^2}\sum_{k\neq i}\frac{\partial \mathrm{g_{ij}}}{\partial \mathrm{A}} = \frac{-2\sum_{k\neq i} g_{ik}\,A(x_i - x_k)(x_i - x_k)^T}{\sum_{s\neq i} g_{is}}$$

$$= -2p_{ij}A\left(\sum_{k\neq i} p_{ik}\,(x_i - x_k)(x_i - x_k)^T\right)$$

代入上两式，我们有：

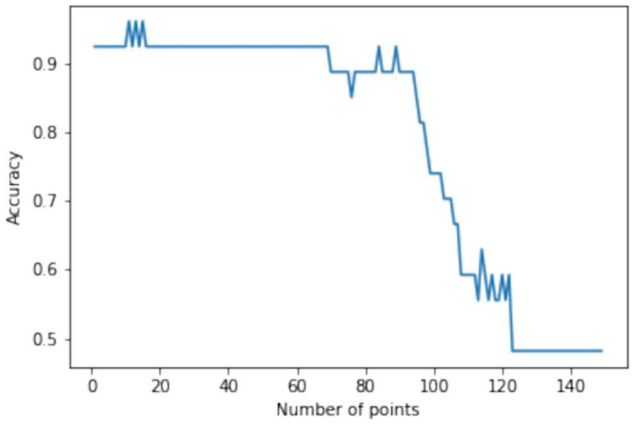
$$\frac{\partial p_{ij}}{\partial A} = -2p_{ij}A\left((x_i - x_j)(x_i - x_j)^T - \sum_{k\neq i} p_{ik}(x_i - x_k)(x_i - x_k\}^T\right)$$

$$\frac{\partial f}{\partial A} = -2A\sum_{i=1}^{N}\sum_{j\in C_i} p_{ij}\left((x_i - x_j)(x_i - x_j)^T - \sum_{k\neq i} p_{ik}(x_i - x_k)(x_i - x_k\}^T\right)$$

7. 实验结果与分析

3.1 欧氏距离

在训练集上的准确率随 K 值的变化曲线如下图所示。



图表 6:欧式距离程序中 K 和准确率

在 k 等于 11 时准确率取到最大值 96%。

由实验可知，K 值越小，则模型复杂度较高，容易发生过拟合，同时预测结果对近邻的实例点非常敏感。

K 值越大可以减少学习的 various，但是学习的 bias 会增大，模型的复杂度会下降。但对于小型数据而言，过大的 K 会导致大量样本被错分为数量最多的那一类。

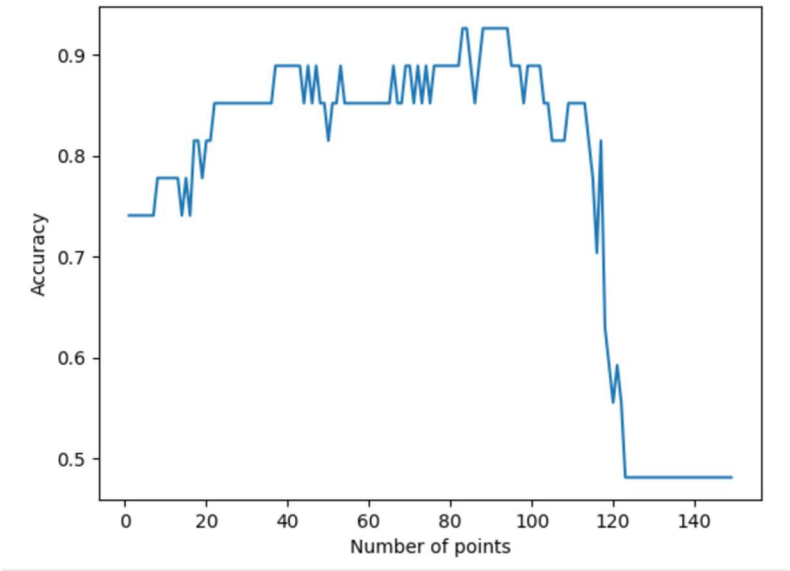
在 k 达到 100 以上时算法已经失去意义，因为几乎所有的点都被判定为数据最多的类。

3.2 马氏距离

用梯度下降法对 A 进行更新，在第 100 次更新的结果为：

```
 [[5.08443656e-02,  5.12681069e-03,  6.06616156e-03, -4.32789149e+00,
  2.68310777e-03,  3.75712687e-01,  3.58570518e-02 , 2.27764534e-03,
  1.39429652e-01],
[-2.26788622e+00 ,-1.34528932e+00, -5.10608630e-02, -3.43686243e-01,
  7.17562172e-02 ,-3.28447762e-01, -7.10408191e-03, -4.81660109e-02,
  2.75824997e-02, -4.20081216e+00, -1.76492057e-03,  1.30272212e-02,
  4.48312757e-02]]
```

准确率随 K 变化的曲线图为：



图表 7：马氏距离中 K 值和准确率

3.3 在测试集上验证结果

实验要求在测试集上进行测试，由于没有标签，可以使用两种算法（欧式和马氏）同时分类，比较结果的差异性如何。

```
[1, 3, 1, 2, 2, 1, 1, 1, 3, 1, 2, 2, 1, 1, 2, 2, 2, 2, 3, 2, 2, 1, 1, 2, 2, 1]
```

图表 8：欧式距离结果

```
[1, 3, 1, 2, 2, 1, 1, 1, 3, 1, 2, 2, 1, 1, 2, 2, 2, 2, 3, 2, 2, 1, 1, 2, 2, 1]
```

图表 9：马氏距离结果

观察到两种算法结果一样，说明算法准确率较高。

1. 代码附录

```
"""
使用欧式距离做 KNN 分类
输出正确率验证结果

61518407 李浩瑞 2021.1.5
"""
import csv
import random
import os
import sys
import operator
```

```
import math
import numpy as np
import matplotlib.pyplot as plt
import scipy.io as scio

project_path = os.getcwd()   # 获取当前文件路径的上一级目录
train_path = project_path+r"\KNN\data\train_data.mat"  # 拼接训练路径字符串
test_path = project_path+r"\KNN\data\test_data.mat"  # 拼接测试路径字符串
val_path = project_path+r"\KNN\data\val_data.mat"


data = scio.loadmat(train_path)['data']
label = scio.loadmat(train_path)['label']
valdata = scio.loadmat(val_path)['data']
vallabel = scio.loadmat(val_path)['label']


def fullMean(data):
    """
    获得整个数据的均值
    """
    result = []
    sums = [0]*len(data[0])
    count = [0]*len(data[0])
    for j in range(len(data[0])):
        for i in range(len(data)):
            if(np.isnan(data[i][j]) == False):
                sums[j] = sums[j]+data[i][j]
                count[j] = count[j]+1
    for i in range(len(sums)):
        temp = sums[i]/count[i]
        result.append(temp)
    return result


def rowMean(data, label):
    """
    获得每个向量的均值
    """
    result = np.zeros((3, len(data[0])))
    count = np.zeros((3, len(data[0])))
    for i in range(len(data)):
        for j in range(len(data[i])):
            if(np.isnan(data[i][j]) == False):
```

```
                temp = label[i]-1
                result[temp[0]][j] = result[temp[0]][j]+data[i][j]
                count[temp[0]][j] = count[temp[0]][j]+1
    for i in range(len(result)):
        for j in range(len(result[i])):result[i][j] = result[i][j]/coun
t[i][j]
    return result


def fillNan(data, label):
    mean = rowMean(data, label)
    for i in range(len(data)):
        for j in range(len(data[i])):
            if(np.isnan(data[i][j])):
                temp = label[i]
                data[i][j] = mean[temp[0]-1][j]
    return data


def Normalize(data, mean, var):
    for i in range(len(data)):
        for j in range(len(data[i])):data[i][j] = (data[i][j]-
mean[j])/pow(var[j], 0.5)


def fullVar(data):
    """
    获得整个数据的方差
    """
    result = []
    mean = fullMean(data)
    sums = [0]*len(data[0])
    for i in range(len(data)):
        for j in range(len(data[0])):
            t = label[i]
            sums[j] += pow(data[i][j]-mean[j], 2)
    for i in range(len(sums)):
        temp = sums[i]/len(data)
        result.append(temp)
    return result


def Fillnan_and_Normal(train, test, label):
    fillNan(train, label)
    mean = fullMean(train)
```

```
    var = fullVar(train)
    Normalize(train, mean, var)
    Normalize(test, mean, var)


Fillnan_and_Normal(data, valdata, label)


def O_distance(testInstance, trainInstance):
    """
    计算欧式距离
    return float
    """
    length = len(testInstance)
    distance = 0
    for i in range(length):
        testInstance1 = float(testInstance[i])
        trainInstance1 = float(trainInstance[i])
        distance += (np.square(testInstance1-trainInstance1))
    return math.sqrt(distance)


def M_distance(A, x1, x2):
    """
    调用欧式距离函数计算马氏距离
    return float
    """
    return O_distance(np.dot(A, x1), np.dot(A, x2))


def Prob(data, A):
    """
    计算概率矩阵
    return [len(data),len(data)]
    """
    n = len(data)
    P = np.zeros((n, n))
    for i in range(n):
        sums = 0
        for j in range(n):
            if j == i:
                P[i][j] = 0
            else:
                for k in range(n):
                    if k != i:
```

```
                        sums = sums + \
                            np.exp(-
np.square(M_distance(A, data[i], data[k])))
                    else:
                        pass
                P[i][j] = np.exp(-
np.square(M_distance(A, data[i], data[j])))
    return P


def KNN(k):
    """
    欧式距离 KNN
    """
    r = 0
    w = 0
    for i in range(len(valdata)):
        distance = []
        pred_label = []
        for j in range(len(data)):
            D = O_distance(valdata[i], data[j])
            distance.append((label[j], D))
        distance.sort(key=operator.itemgetter(1))
        temp = distance[0:k]
        for klabel in temp:pred_label.append(int(klabel[0]))
        pred = max(pred_label, key=pred_label.count)
        if pred == vallabel[i]:r += 1
        else:w += 1
    acc = r/(r+w)  # 返回正确率用于绘图
    return acc

if __name__ == '__main__':
    x = range(1, 150)
    y = []
    for i in x:
        y.append(KNN(i))
    plt.plot(x, y)
    plt.xlabel('Number of points')
    plt.ylabel('Accuracy')
    plt.savefig('knn_1.png')

"""
使用马式距离做 KNN 分类
输出正确率验证结果
```

```
61518407 李浩瑞 2021.1.5
"""
from KNN_O_407 import *


def Grad(data, label, A):
    y = len(data[0])
    n = len(data)
    Prow = []
    """计算概率矩阵"""
    P = np.zeros((n, n))
    for i in range(n):
        sums = 0
        for j in range(n):
            if j == i:
                P[i][j] = 0
            else:
                for k in range(n):
                    if k != i:
                        sums = sums + \
                            np.exp(-
np.square(M_distance(A, data[i], data[k])))
                    else:
                        pass
                P[i][j] = np.exp(-
np.square(M_distance(A, data[i], data[j])))
    """初始化"""
    for i in range(len(P)):
        temp = 0
        for j in range(len(P[0])):
            if label[i] == label[j]:
                temp = temp+P[i][j]
        Prow.append(temp)
    """求梯度"""
    gradients = np.zeros((y, y))
    for i in range(len(data)):
        k_sum = np.zeros((y, y))
        k_same_sum = np.zeros((y, y))
        for k in range(len(data)):
            out_prod = np.outer(data[i] - data[k], data[i] - data[k])
            k_sum += P[i][k] * out_prod
            if label[k] == label[i]:k_same_sum += P[i][k] * out_prod
        gradients += Prow[i] * k_sum - k_same_sum
    gradients = 2 * np.dot(A, gradients)
    return gradients
```

```
def calA(data, label, n, lr=0.01, epoch=100):
    y = len(data[0])
    A = np.random.standard_normal(size=(n, y))
    for i in range(epoch):
        grad = Grad(data, label, A)
        if (np.max(grad) < 3):break
        A = A-grad*A*lr
        print("epoch:{},A:{}".format(i, A))
    return A


#A=calA(data,label,2,0.01, 100)
# 使用上式计算出最优的 A
A = [[4.89514866e-02,  9.67910746e-02, -1.23441573e+00, -
2.08268957e+00,
      5.08443656e-02,  5.12681069e-03,  6.06616156e-03, -
4.32789149e+00,
      2.68310777e-03,  3.75712687e-01,  3.58570518e-02, 2.27764534e-03,
      1.39429652e-01],
     [-2.26788622e+00, -1.34528932e+00, -5.10608630e-02, -3.43686243e-
01,
      7.17562172e-02, -3.28447762e-01, -7.10408191e-03, -4.81660109e-
02,
      2.75824997e-02, -4.20081216e+00, -1.76492057e-03,  1.30272212e-
02,
      4.48312757e-02]]


def M_KNN(k):
    r = 0
    w = 0
    for i in range(len(valdata)):
        distance = []
        pred_label = []
        for j in range(len(data)):
            dist = M_distance(A, valdata[i], data[j])
            distance.append((label[j], dist))
        distance.sort(key=operator.itemgetter(1))
        temp = distance[0:k]
        for klabel in temp:pred_label.append(int(klabel[0]))
        pred = max(pred_label, key=pred_label.count)
        if pred == vallabel[i]:r += 1
        else:w += 1
```

```
    acc = r/(r+w)  # 返回正确率用于绘图
    return acc


if __name__ == '__main__':
    x = range(1, 150)
    y = []
    for i in x:
        y.append(M_KNN(i))
    plt.plot(x, y)
    plt.xlabel('Number of points')
    plt.ylabel('Accuracy')
    plt.savefig('knn_2.png')
```

课程建议

应薛老师要求，最后给这门课提一些小小的建议：建议提前布置大作业内容，在最后几周忙于准备考试，压力很大。如果可以在教学过程中，比如每 4 周布置一份大作业，会减轻很多压力。

心得体会

首先要感谢薛老师耐心讲授的知识，还要感谢助教们的认真负责，多亏了从老师不断学习知识，在实验不断反思讨论，最终三项实验的正确率和想挖掘测试的东西都完成了。

实验过程历经艰难险阻，有痛苦于求逆问题无法解决的夜晚，也有设置动态学习率之后看到迅速收敛的欣喜。总的来说把课上的知识点用在真实项目中并看到代码一步一步起作用真的很有成绩感，在这次大作业里也学到很多。

最后还想感谢和我一起奋战到多个深夜的同学们，虽然我们

选题不同，解决问题的思路不同，在交流想法时还多有争吵，很多求解的灵感正是来源于一起对一个问题苦思冥想时候的火花，比如通过伪逆避开大量的零值等。

本次实验虽然花费了大量的时间，但最终无论是正确率还是想探索的东西都没有留下遗憾，非常感谢能有这次实验的机会。