**人工智能基础LAB2实验**

**王湘峰 PB19030861**

1. **实验内容与提示**

本次实验包含传统机器学习与深度学习两部分。实验部分需要使⽤python=3.6，建议使⽤anaconda管理python环境，深度学习部分要求使⽤pytorch=1.8.1，torchvision=0.9.1完成（安装说明⻅ <https://pytorch.org> ，学习教程可以参考[PyTorch官方教程中文版 (pytorch123.com)](https://pytorch123.com/)，实验部分使⽤CPU⾜够训练，如果想体验GPU的速度可以使⽤colab。

1. **传统机器学习**

**2.1线性分类器**

（1）对引⼊了 L2 规范化项之后的最⼩⼆乘分类问题进⾏推导。即求解以下优化问题：

(2)基于(1)的结果，实现linearClassification.py中未完成的代码部分。

由数学知识得，为了求上式的最小值，可以通过对w求梯度然后不断通过迭代来找到最佳的w，即：

其中

当 < 时停止迭代

**运行结果如下：**

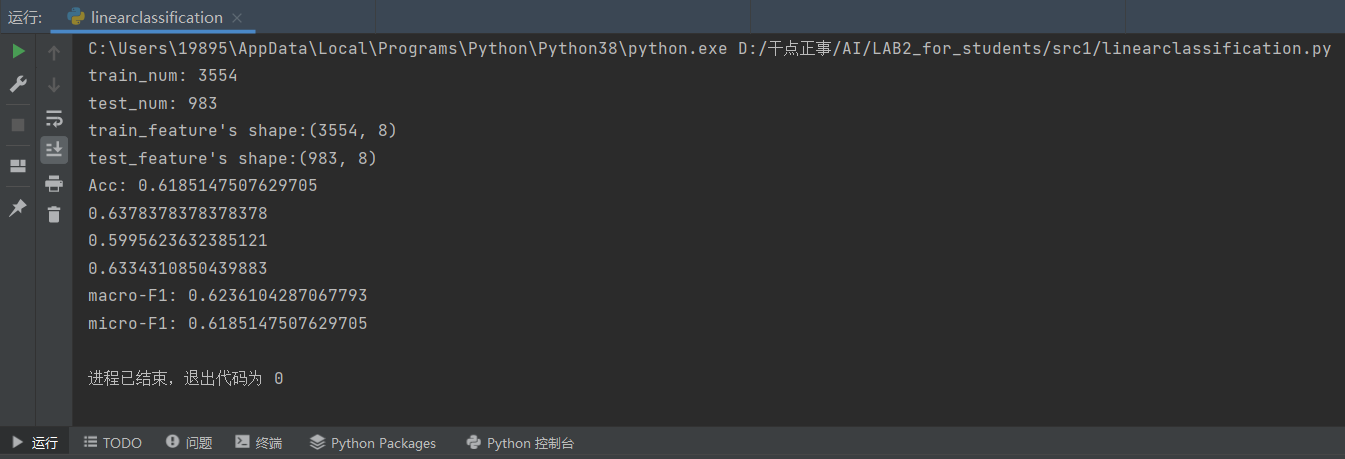


图1：LinearClassifier

线性分类器的准确率为61.85%

**试错与总结**

起初在编写代码的时候直接将每个类别的误差相加而未取平均，导致分类效果不理想，仔细检查错误后予以改正，将准确率由40%提高到了60%。

出现该失误的主要原因是没有深刻理解算法的含义。

**2.2 朴素贝叶斯分类器**

完善nBayesClassifier.py的代码,以实现朴素⻉叶斯分类器，使⽤拉普拉斯平滑计算条件概率和先验概率。

其中D表示训练集,为类别为c的数据，表示类别为c，第i个属性为x的数据，表示第i个属性的可能取值，判定准则为：

由于属性中大部分为连续性属性，所以需要进行处理，通常处理的方法有两种：

1. 将连续属性离散化，用相应的离散区间代替连续属性
2. 假设连续型变量符合某种概率分布，通过训练数据估计分布的参数，通常使用高斯分布来估计，即估计每个类的均值和方差

本次试验采用了第二种方法，通过numpy库自带的np.mean()和np.var()函数求得每个属性在条件为c的情况下的分布函数，并将条件概率的计算公式替代为

对离散型变量进行统计时用到了Counter库，直接得到了离散变量各取值的个数。

对于变量参数的存储方式，离散型变量的字典结构是｛｝，label以及训练集的第一个属性都采用这种格式存储（第一个属性新建了名为P1x的字典）；

连续型变量的存储结构是｛(i,j):(,)｝，它表示当标签为j时第i个(连续)属性的分布参数。

**最后运行的结果如下：**

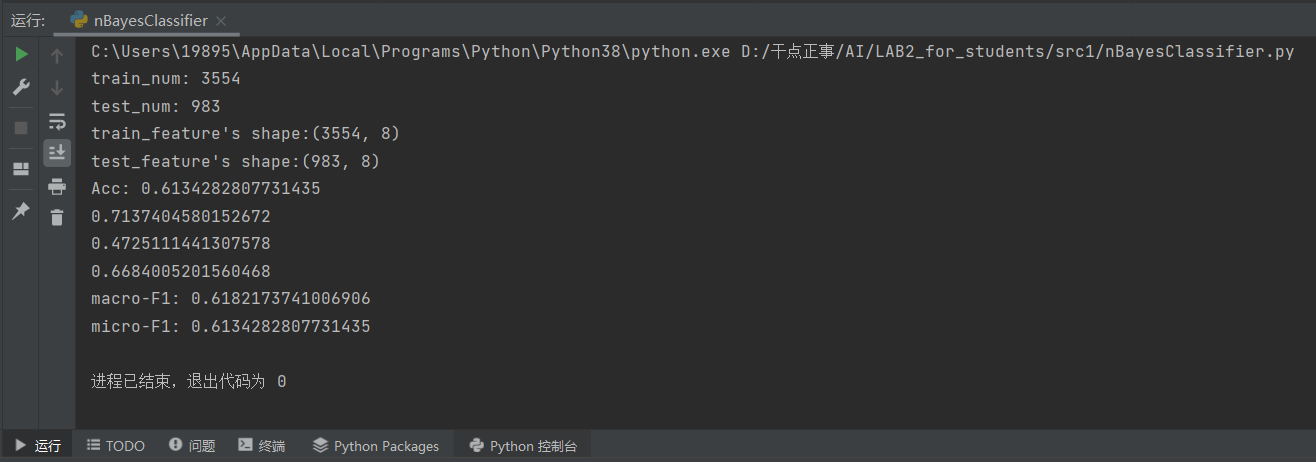


图2：nBayesClassifier

预测准确率为61.34%

* 1. **SVM分类器**

完善SVM.py中的代码，以实现⽀持软间隔与核函数的SVM。

对于K分类(K>2)，我们使⽤one-vs-all策略训练，具体为：对于任⼀类别，我们将其看作正类“1”，其余类别看作负类“-1”, 分别训练得到K个⼆分类器；测试时，对于⼀给定样本，分别计算该样本在K个⼆分类器上的输出/分数，取最⼤输出/分数所对应的分类器的正类作为最终的预测类别。（这⼀部分已在代码中给出）。

**注意：**

1. 分类器函数返回值应为SVM预测的分数，即

而非

对于实现多种核的SVM，构造核矩阵是一个较佳的方案，即计算矩阵

其中，其中指训练集第i行的数据()，根据kernel的类别分别可分为：

1. Gauss核：
2. Linear核：
3. Poly核：

之后将主要计算支持向量SV以及偏置***b***

由于支持向量机参数的求解可转化为凸优化问题，并且可以调用现有的凸优化库，因此只需要计算出对应的凸优化问题的矩阵即可。

凸优化问题的一般形式为：

根据要求可知：

x为n个点的拉格朗日乘子，理论上支持向量的不为0，其余为0

P=Kernel, q=ones(n,1)，其中n为训练集数量；

当软间隔为C时，G为 Diag[-1,……-1](括号中为n个-1)与n阶单位阵的垂直拼接，h为n阶**0**矩阵与Diag[C,……,C](n个C)的水平拼接

A为label的列向量形式

b为

由于计算的机器误差等原因，我们认为的点为支持向量。

***b*** =

***prediction*** =

**程序运行结果如下：**

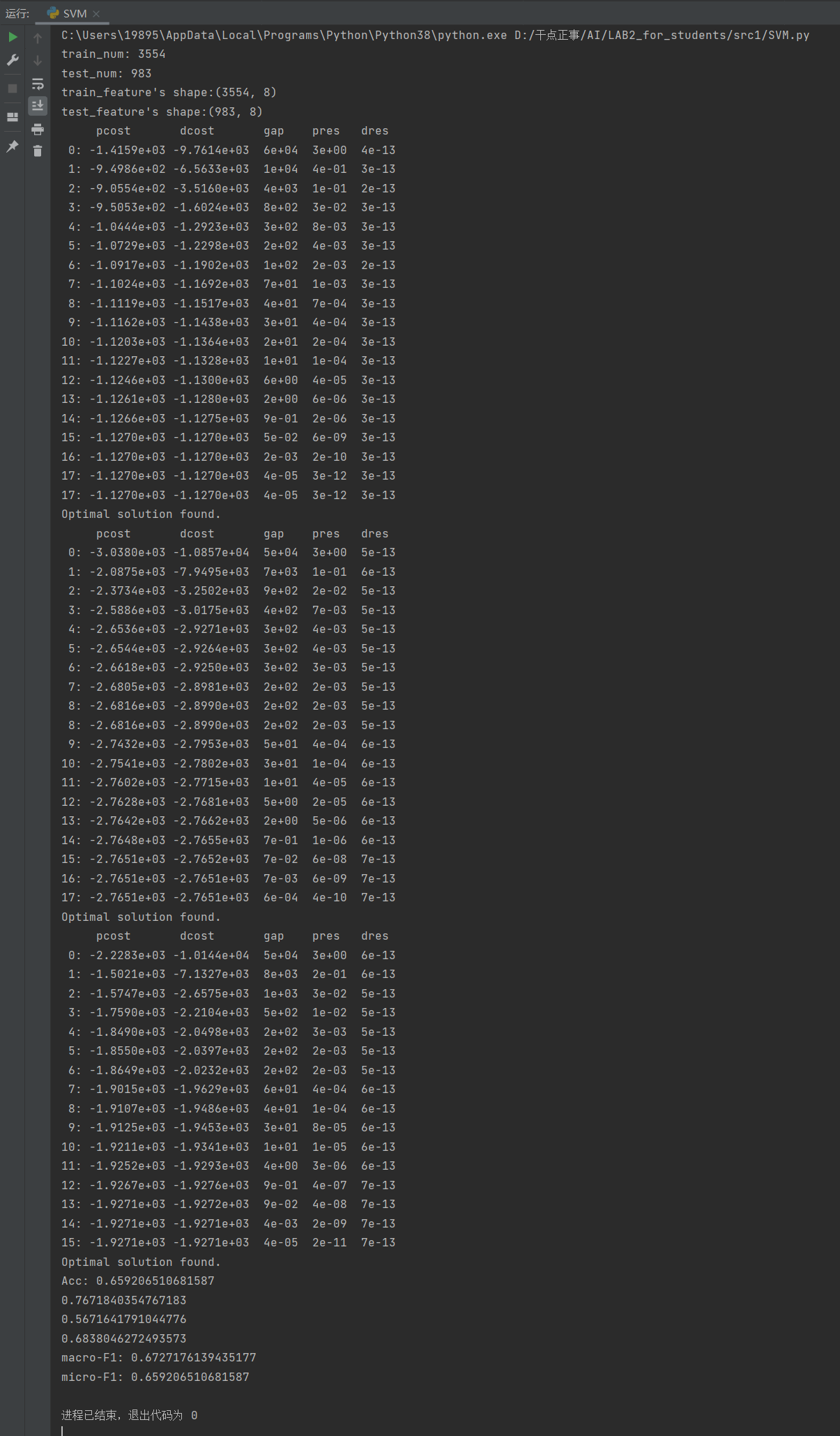


图3：Linear核

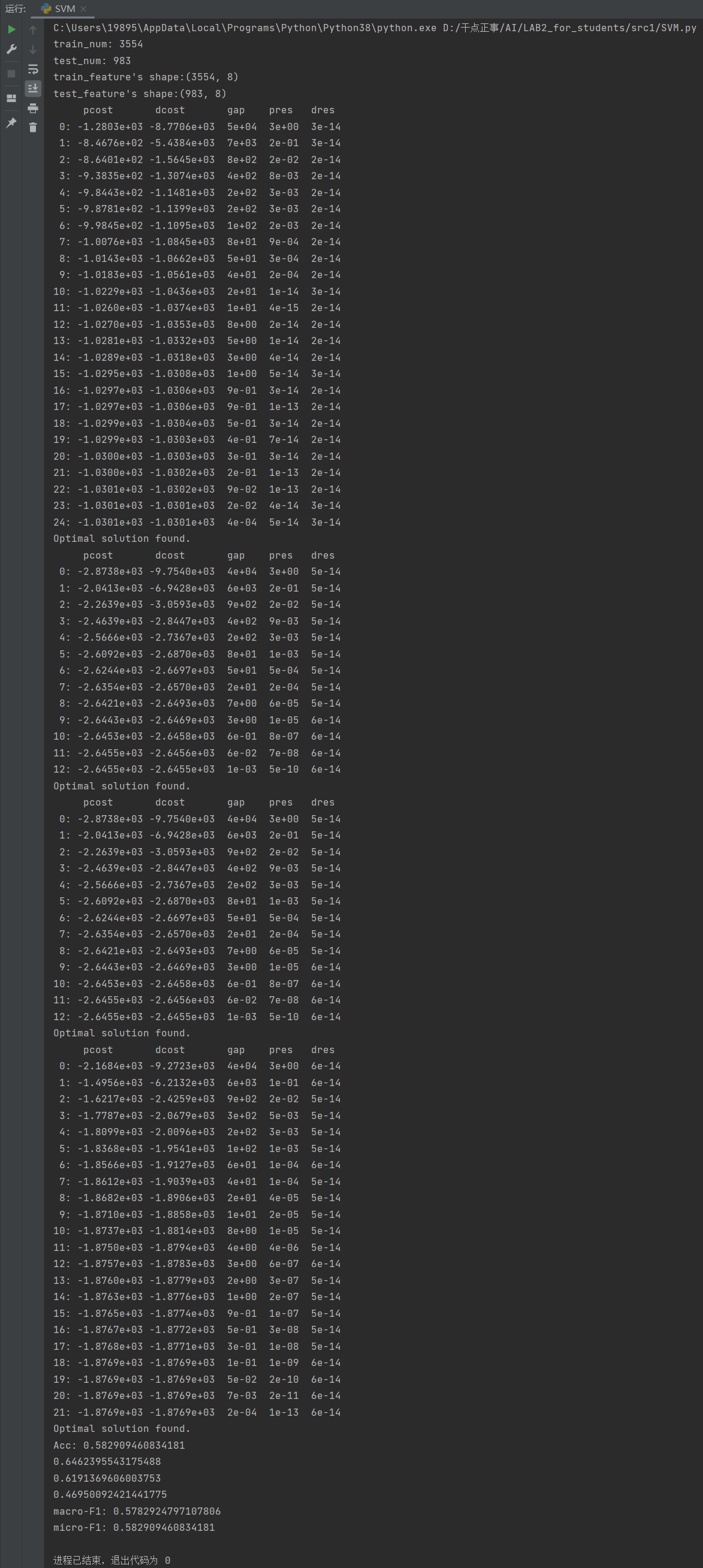


图4：Gauss核

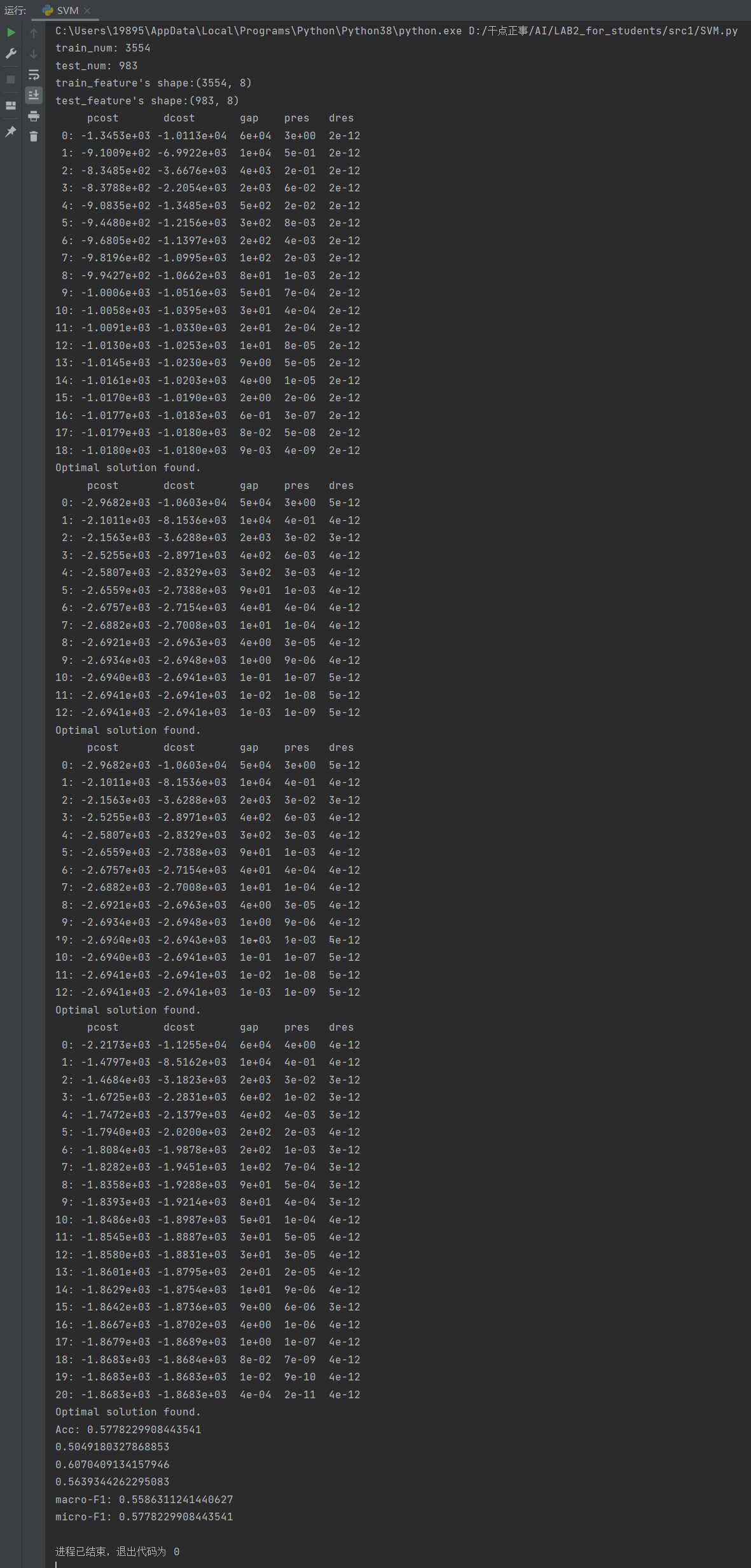


图5：Poly核

线性核预测准确率为65.92%

高斯核预测准确率为58.29%

多项式核预测准确率为57.78%

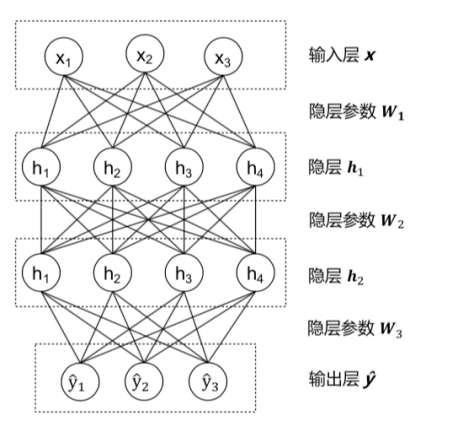
通过比较发现，三种核预测的准确率差别并不大，对于具体问题不同的核的表现也不尽相同，但是在计算速度上线性核的速度明显快于其他核，在实际问题中如果线性核的效果较好，则直接使用线性核足矣。

1. **深度学习**

**3.1手写感知机模型并进⾏反向传播**

实验内容：实现⼀个4层的感知机模型（**隐层神经元设置为5，4，4，3**，即输⼊的特征尾为5，输出的类别个数的3，激活函数设置为**sigmoid**）；实现BP算法；实现梯度下降算法。

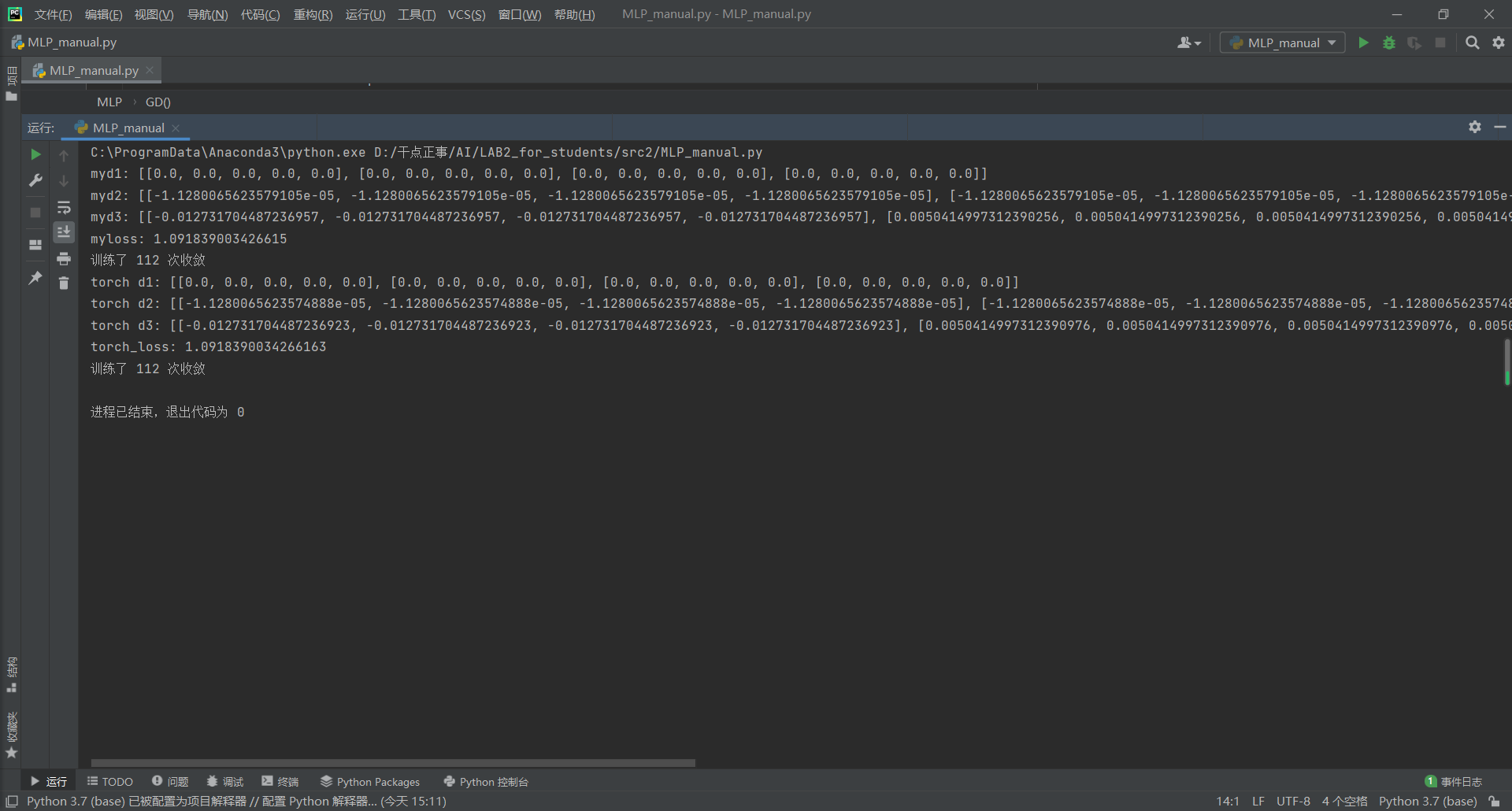
实验要求：通过矩阵运算实现模型；实现各参数的梯度计算，给出各参数矩阵的梯度，并与pytorch⾃动计算的梯度进⾏对⽐；实现梯度下降算法优化参数矩阵，给出loss的训练曲线。



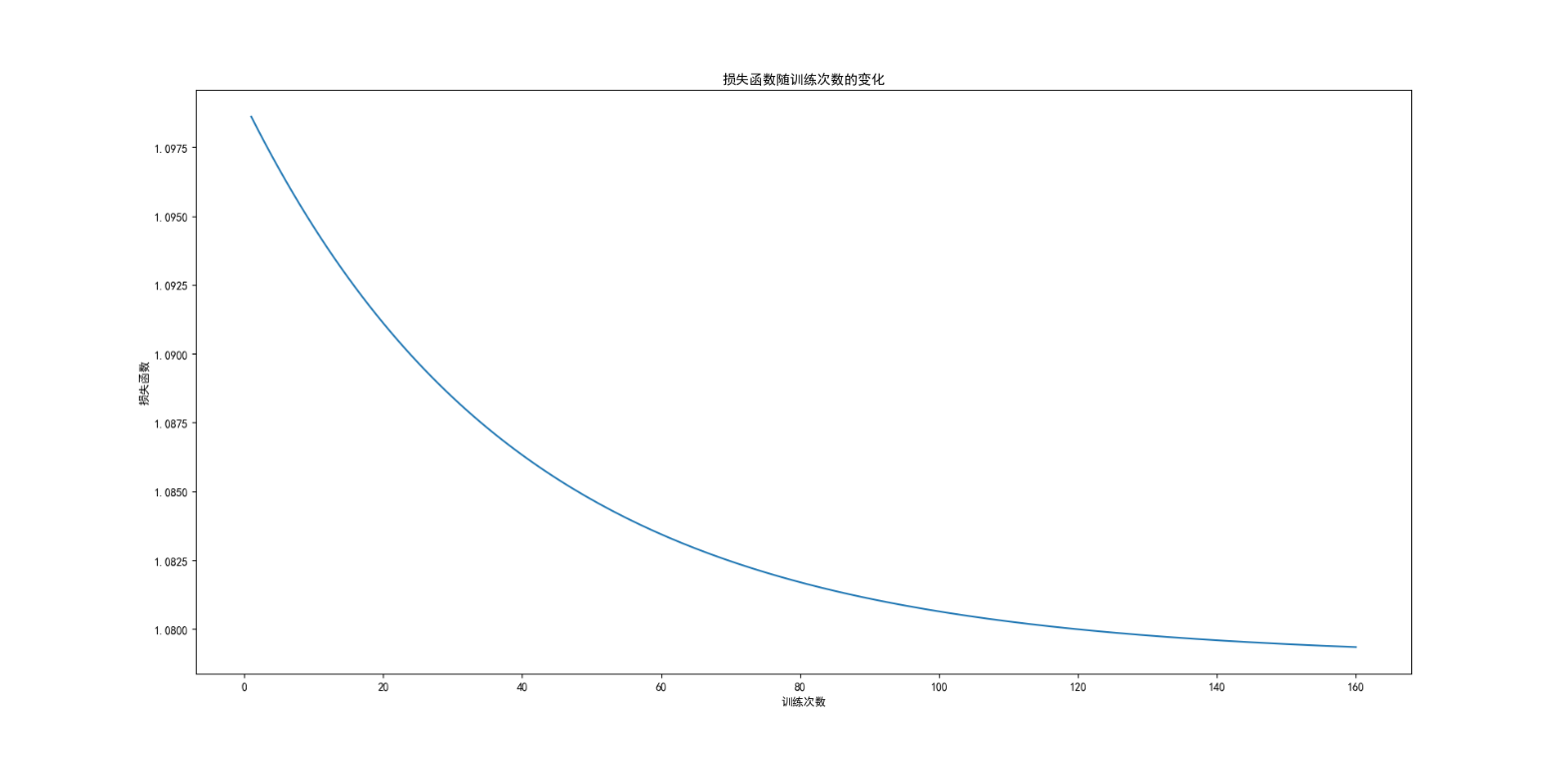
我算法的大致步骤是：

1. 初始化W1,W2,W3
2. 前向传播得到预测结果
3. 依据矩阵求导公式计算W1,W2,W3的梯度，更新矩阵
4. 若，则算法终止，否则回到(2)

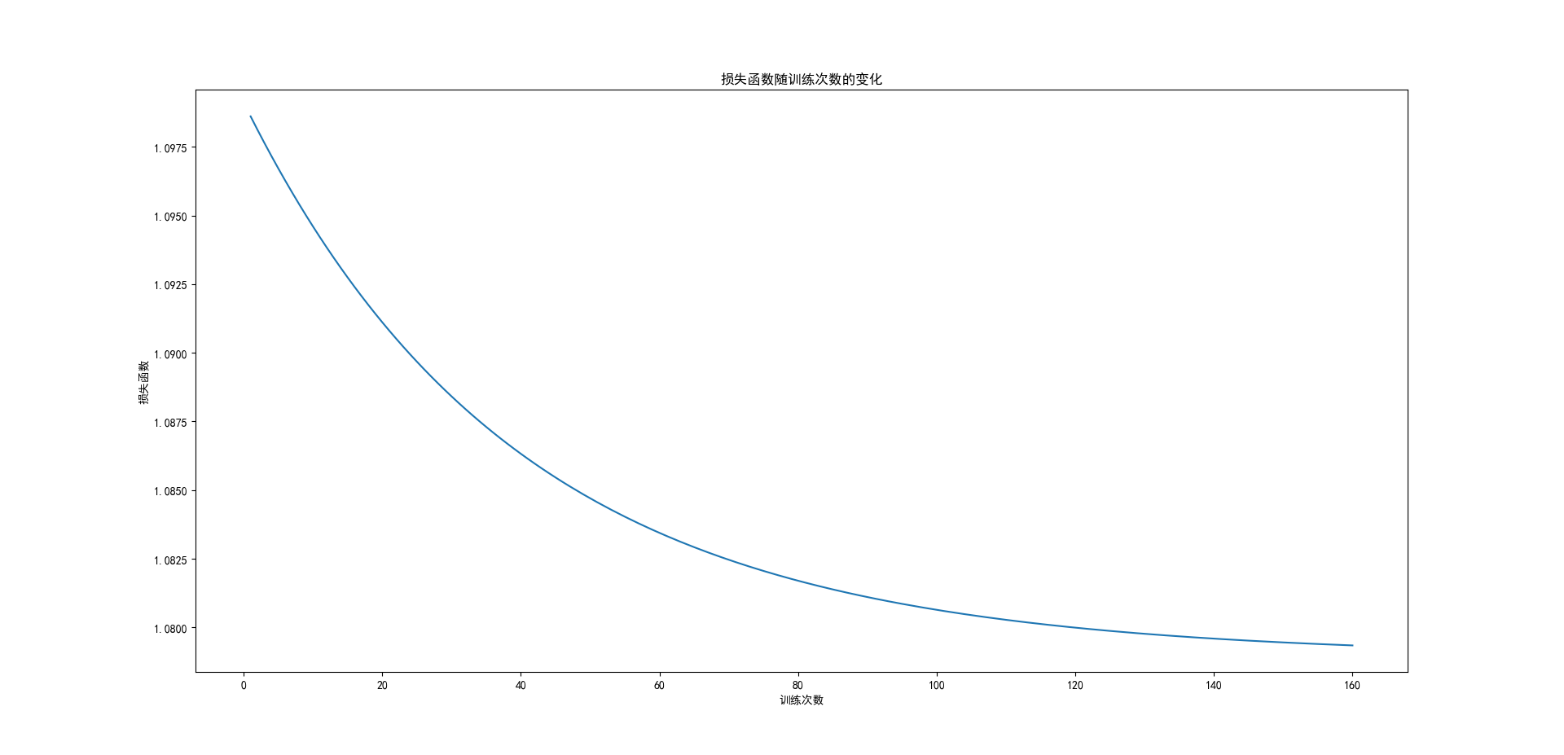
**实验结果如下：**



图为训练完成时W1,W2,W3的梯度矩阵、训练次数和交叉熵，以及与torch自动求导得到的结果进行对比。



手写MLP得到的loss曲线



Torch得到的loss曲线

实验结果表明，二者的梯度矩阵与交叉熵loss完全一致，保证了程序的可靠性。此外由于数据是随机生成的，因此本质上并无规律可言，因此优化的效果不明显。

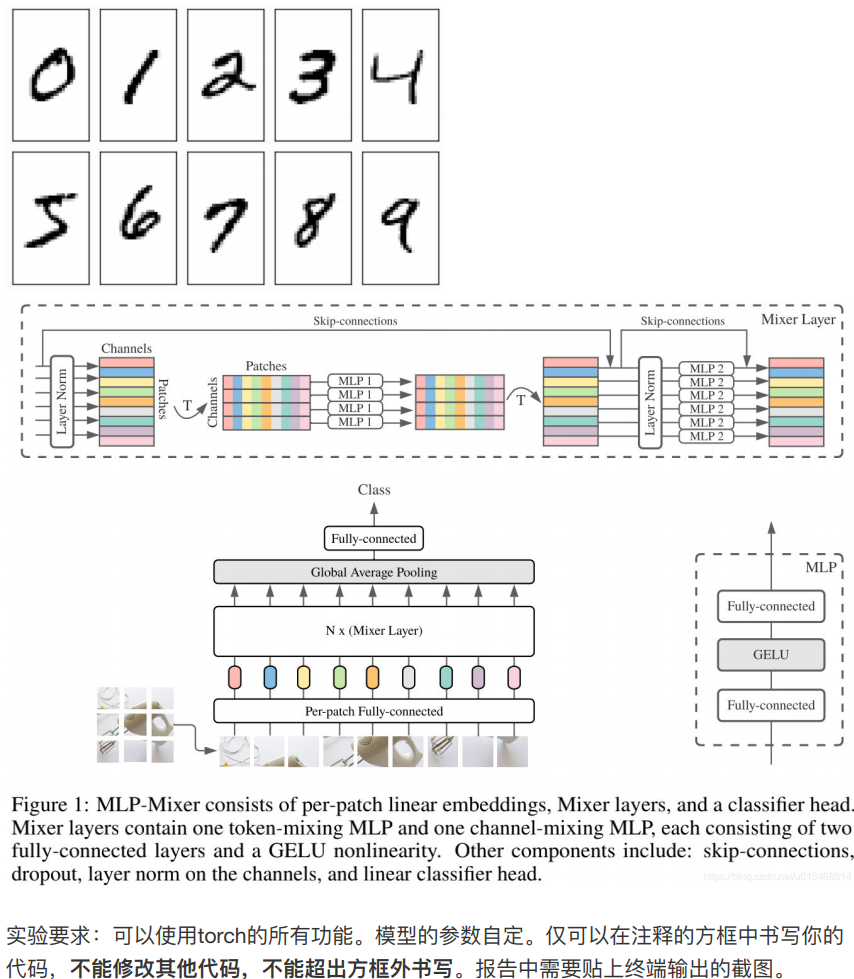
**小结:**

这个实验在完成时遇到了数不胜数的小困难，比如对矩阵求导公式的误解、矩阵维数的不对应，陌生的torch语法等等……为了解决这些问题，我通过从头对反向传播公式进行推导，逐渐理解了其中的含义；以及善用搜索引擎解决了torch的语法问题，最终实现了手写MLP的任务。这个实验对我的搜索能力有了较大的提高，对矩阵的链式求导有了更深的理解和掌握。

**3.2复现MLP-Mixer**

**实验内容：**复现MLP-Mixer模型，并在MNIST数据集上进⾏测试。

数据集介绍：数据集由60000⾏的训练数据集（trainset）和10000⾏的测试数据集（testset）组成，包含从0到9的⼿写数字图⽚，如下图所示，分辨率为28\*28。每⼀个MNIST数据单元有两部分组成：⼀张包含⼿写数字的图⽚和⼀个对应的标签（对应代码⽂件中的data和target）



**实验大致原理为**：

先将输入图片拆分成patches，然后通过Per-patch Fully-connected将每个patch转换成feature embedding，然后送入N个Mixer Layer，最后通过Fully-connected进行分类。

MLP\_Mixer 采用了token-mixing MLP 和 channel-mixing MLP两种不同的MLP层。其中token-mixing MLP 负责不同空间位置的token之间进行通信；而channel-mixing MLP负责不同channel的token之间进行通信。token-mixing MLP block作用在每个patches的列上，即先对patches部分进行转置，并且所有列参数共享MLP1，得到的输出再重新转置一下。channel-mixing MLP block作用在每个patche的行上，所有行参数共享MLP2。这两种类型的层交替执行以促进两个维度间的信息交互。

由于本次实验只有两个类(**class**)，因此我将MLP Block合并到了MLP-Layer当中。其中mlp1代表**token**-mixing block，mlp2代表**channel**-mixing block。

**神经网络结构如下：**

**嵌入层：**

初始化，将输入图片分解为N个SS个小图片(patches)，初始化神经网络权重。将N个图片经过全连接(fully-connected)映射到S个C维向量(token)。

**Mixer层：**

1. 先将SxC的输入对通道进行层归一化（对单个样本（token）的所有通道求均值和方差，然后归一）
2. 其次经转置后CxS经token-mixing块，对相同通道的不同token进行空间上的混合（此处输入是对位置敏感的，所以不需要transformer的位置嵌入）
3. 然后转置为SxC后与原始输入x进行跨层直连得到x2
4. 最后经层归一化后，用channel-mixing对相同位置的不同通道进行混合，得到SxC的输出与x2跨层直连得到SxC的最终输出。

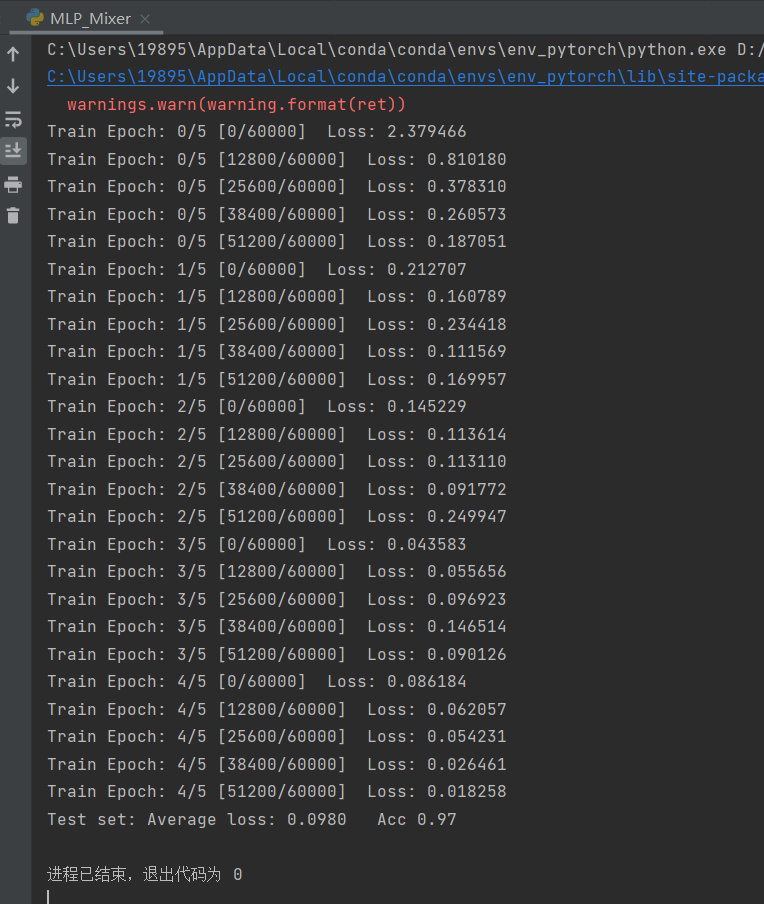
**输出层：**

单层全连接，输入是SxC经全局平均池化（对各个channel求平均）后C维的向量，输出是n维向量，n是类别个数。

**优化器**：

预测的损失由交叉熵(CrossEntropy)计算，采用现成的优化器库函数实现，同时计算预测准确率与训练平均交叉熵。

**实验结果如下：**



由于处理时防止过拟合加入了正态误差，因此每次预测的结果都是具有一定的随机性的。

经过多次调参以及原论文的建议，选择了Adamax优化器， 2x2的patches，384个隐层，2层的depth，预测准确率大致为97%~98%.

该实验实现较为顺利，而且有了前四次的实战练习，并无太大障碍。

**实验总结**

本次实验涵盖面广，难度由易到难，循序渐进，既考验代码功底，又考察对理论知识的理解和推导。总体上实验花费了不小的精力和智慧，同时收获也颇丰。尽管实验前前后后遇到无数大大小小的困难和bug，但是最终还是坚持了下来，一点一点突破障碍，最终完成了实验。主观感受上这次实验几乎等同于将主流机器学习(Linear、NaiveBayes、SVM、MLP等)又重新学习了一遍，对这些算法的理解又更深入了一步，以及复现Google今年的新成果，与时俱进，提高了实际解决问题的能力，同时锻炼了耐心与意志，我认为这是一次很有成果和成就感的修炼。为此感谢老师和助教对本次实验的精心设计！