

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（安装说明见 <https://pytorch.org>，学习教程可以参考 [PyTorch 官方教程中文版 \(pytorch123.com\)](#)，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。

2. 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：

$$\min_w (Xw - y)^2 + \lambda ||w||^2$$

(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：

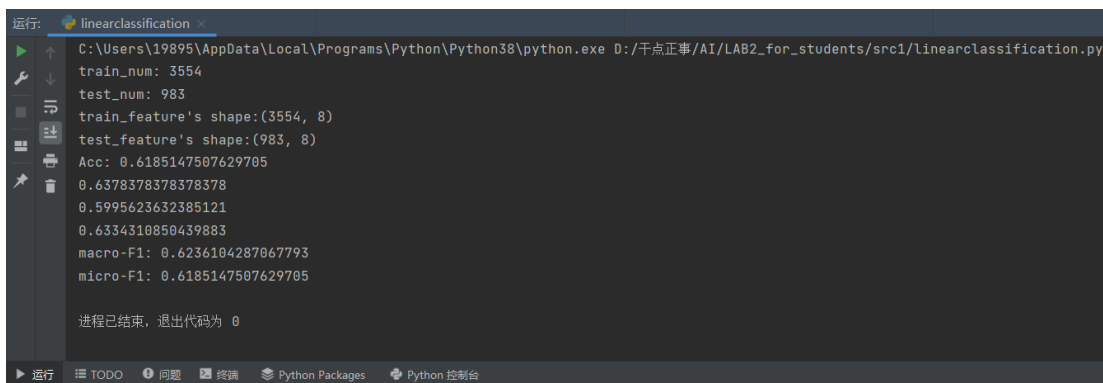
$$\frac{\partial J(w)}{\partial w_i} = 2X_i^T(\hat{y} - y) + 2\lambda w_i$$

其中 $\hat{y} = Xw$

$$W^{(k+1)} = W^{(k)} - \alpha \frac{\partial J(W)}{\partial W}$$

当 $|\hat{y} - y| < \varepsilon$ 时停止迭代

运行结果如下：



```
运行: linearclassification
C:\Users\19895\AppData\Local\Programs\Python\Python38\python.exe D:/干点正事/AI/LAB2_for_students/src1/linearclassification.py
train_num: 3554
test_num: 983
train_feature's shape:(3554, 8)
test_feature's shape:(983, 8)
Acc: 0.6185147507629705
0.6378378378378378
0.5995623632385121
0.6334310850439883
macro-F1: 0.6236104287067793
micro-F1: 0.6185147507629705

进程已结束,退出代码为 0
```

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, 以实现朴素贝叶斯分类器, 使用拉普拉斯平滑计算条件概率和先验概率。

$$P(c) = \frac{|D_c| + 1}{|D| + N}$$
$$P(x_i|c) = \frac{|D_{c,x_i}| + 1}{|D| + N_i}$$

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

$$h_{nb}(x) = \underset{c \in Y}{\operatorname{argmax}} P(c) \prod_{i=1}^d P(x_i|c)$$

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 μ 和方差 σ^2

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

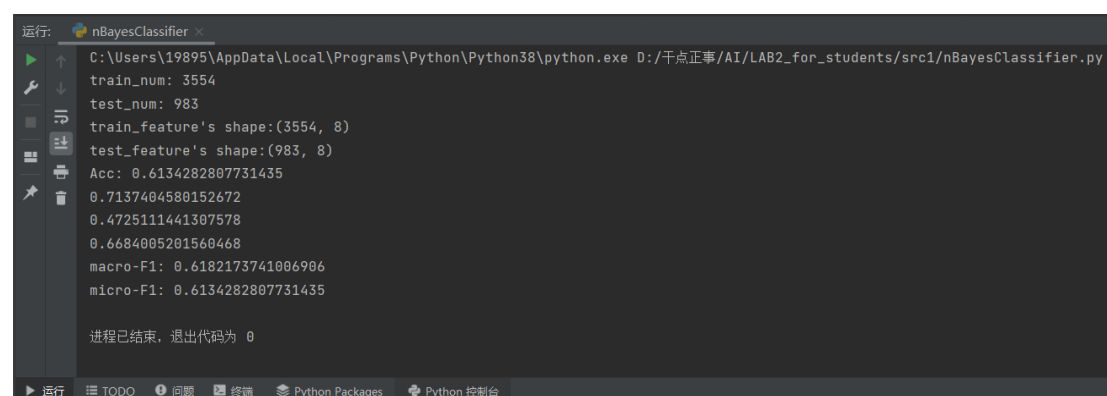
求得每个属性在条件为 c 的情况下的分布函数，并将条件概率的计算公式替代为

$$P(x_i|c) = \frac{1}{\sqrt{2\pi\sigma_i^2}} e^{-\frac{(x_i-\mu_i)^2}{2\sigma_i^2}}$$

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 $\{x_i : p_i\}$ ，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 $\{(i, j) : (\mu, \sigma^2)\}$ ，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：



```
运行: nBayesClassifier
C:\Users\19895\AppData\Local\Programs\Python\Python38\python.exe D:/干点正事/AI/LAB2_for_students/src1/nBayesClassifier.py
train_num: 3554
test_num: 983
train_feature's shape:(3554, 8)
test_feature's shape:(983, 8)
Acc: 0.6134282807731435
0.7137404580152672
0.4725111441307578
0.6684005201560468
macro-F1: 0.6182173741006906
micro-F1: 0.6134282807731435

进程已结束，退出代码为 0
```

图 2: nBayesClassifier

预测准确率为 61.34%

2.3 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

$$\mathbf{y} = \mathbf{w}\mathbf{x} + \mathbf{b}$$

而非

$$\mathbf{y} = \text{sign}(\mathbf{w}\mathbf{x} + \mathbf{b})$$

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 $K_{(n \times n)}$

其中 $K_{(i,j)} = \text{kernel}(x_i, x_j)$ ，其中 x_i 指训练集第 i 行的数据 ($1 \times n$)，根据 kernel 的类别分别可分为：

1. Gauss 核: $K = e^{-\frac{\sum (x_{(i)} - x_{(j)})^2}{2}}$

2. Linear 核: $K = \mathbf{x}_i \mathbf{x}_j^T$

3. Poly 核: $K = (\mathbf{x}_i \mathbf{x}_j^T)^d$

之后将主要计算支持向量 SV 以及偏置 \mathbf{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：

$$\min_x \mathbf{x}^T \mathbf{P} \mathbf{x} + \mathbf{q}^T \mathbf{x}$$

$$\text{subject to } \mathbf{G} \mathbf{x} \leq \mathbf{h}$$

$$\mathbf{A} \mathbf{x} = \mathbf{b}$$

根据要求可知：

\mathbf{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

$\mathbf{P} = \text{Kernel}$ ， $\mathbf{q} = \text{ones}(n, 1)$ ，其中 n 为训练集数量；

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

\mathbf{A} 为 label 的列向量形式

\mathbf{b} 为 $\mathbf{0}_{(1 \times 1)}$

由于计算的机器误差等原因，我们认为 $\alpha > \varepsilon$ 的点为支持向量。

$$\mathbf{b} = \sum_i \text{label}_{svi} - \alpha_i \text{label}_{svi} \sum_j \text{kernel}(sv_i, sv_j)$$

$$\text{prediction} = \sum_i \alpha_i \text{label}_{svi} \text{kernel}(\text{test}_i, sv)$$

程序运行结果如下：

```
运行: SVM x
C:\Users\19895\AppData\Local\Programs\Python\Python38\python.exe D:/千点正事/AI/LAB2_for_students/src1/SVM.py
train_num: 3554
test_num: 983
train_feature's shape:(3554, 8)
test_feature's shape:(983, 8)

  pcost      dcost      gap      pres      dres
0: -1.4159e+03 -9.7614e+03 6e+04 3e+00 4e-13
1: -9.4986e+02 -6.5633e+03 1e+04 4e-01 3e-13
2: -9.0554e+02 -3.5160e+03 4e+03 1e-01 2e-13
3: -9.5053e+02 -1.6024e+03 8e+02 3e-02 3e-13
4: -1.0444e+03 -1.2923e+03 3e+02 8e-03 3e-13
5: -1.0729e+03 -1.2298e+03 2e+02 4e-03 3e-13
6: -1.0917e+03 -1.1902e+03 1e+02 2e-03 2e-13
7: -1.1024e+03 -1.1692e+03 7e+01 1e-03 3e-13
8: -1.1119e+03 -1.1517e+03 4e+01 7e-04 3e-13
9: -1.1162e+03 -1.1438e+03 3e+01 4e-04 3e-13
10: -1.1203e+03 -1.1364e+03 2e+01 2e-04 3e-13
11: -1.1227e+03 -1.1328e+03 1e+01 1e-04 3e-13
12: -1.1246e+03 -1.1300e+03 6e+00 4e-05 3e-13
13: -1.1261e+03 -1.1280e+03 2e+00 6e-06 3e-13
14: -1.1266e+03 -1.1275e+03 9e-01 2e-06 3e-13
15: -1.1270e+03 -1.1270e+03 5e-02 6e-09 3e-13
16: -1.1270e+03 -1.1270e+03 2e-03 2e-10 3e-13
17: -1.1270e+03 -1.1270e+03 4e-05 3e-12 3e-13
17: -1.1270e+03 -1.1270e+03 4e-05 3e-12 3e-13
Optimal solution found.

  pcost      dcost      gap      pres      dres
0: -3.0380e+03 -1.0857e+04 5e+04 3e+00 5e-13
1: -2.0875e+03 -7.9495e+03 7e+03 1e-01 6e-13
2: -2.3734e+03 -3.2502e+03 9e+02 2e-02 5e-13
3: -2.5886e+03 -3.0175e+03 4e+02 7e-03 5e-13
4: -2.6536e+03 -2.9271e+03 3e+02 4e-03 5e-13
5: -2.6544e+03 -2.9264e+03 3e+02 4e-03 5e-13
6: -2.6618e+03 -2.9250e+03 3e+02 3e-03 5e-13
7: -2.6805e+03 -2.8981e+03 2e+02 2e-03 5e-13
8: -2.6816e+03 -2.8990e+03 2e+02 2e-03 5e-13
8: -2.6816e+03 -2.8990e+03 2e+02 2e-03 5e-13
9: -2.7432e+03 -2.7953e+03 5e+01 4e-04 6e-13
10: -2.7541e+03 -2.7802e+03 3e+01 1e-04 6e-13
11: -2.7602e+03 -2.7715e+03 1e+01 4e-05 6e-13
12: -2.7628e+03 -2.7681e+03 5e+00 2e-05 6e-13
13: -2.7642e+03 -2.7662e+03 2e+00 5e-06 6e-13
14: -2.7648e+03 -2.7655e+03 7e-01 1e-06 6e-13
15: -2.7651e+03 -2.7652e+03 7e-02 6e-08 7e-13
16: -2.7651e+03 -2.7651e+03 7e-03 6e-09 7e-13
17: -2.7651e+03 -2.7651e+03 6e-04 4e-10 7e-13
Optimal solution found.

  pcost      dcost      gap      pres      dres
0: -2.2283e+03 -1.0144e+04 5e+04 3e+00 6e-13
1: -1.5021e+03 -7.1327e+03 8e+03 2e-01 6e-13
2: -1.5747e+03 -2.6575e+03 1e+03 3e-02 5e-13
3: -1.7590e+03 -2.2104e+03 5e+02 1e-02 5e-13
4: -1.8490e+03 -2.0498e+03 2e+02 3e-03 5e-13
5: -1.8550e+03 -2.0397e+03 2e+02 2e-03 5e-13
6: -1.8649e+03 -2.0232e+03 2e+02 2e-03 5e-13
7: -1.9015e+03 -1.9629e+03 6e+01 4e-04 6e-13
8: -1.9107e+03 -1.9486e+03 4e+01 1e-04 6e-13
9: -1.9125e+03 -1.9453e+03 3e+01 8e-05 6e-13
10: -1.9211e+03 -1.9341e+03 1e+01 1e-05 6e-13
11: -1.9252e+03 -1.9293e+03 4e+00 3e-06 6e-13
12: -1.9267e+03 -1.9276e+03 9e-01 4e-07 7e-13
13: -1.9271e+03 -1.9272e+03 9e-02 4e-08 7e-13
14: -1.9271e+03 -1.9271e+03 4e-03 2e-09 7e-13
15: -1.9271e+03 -1.9271e+03 4e-05 2e-11 7e-13
Optimal solution found.
Acc: 0.659206510681587
0.7671840354767183
0.5671641791044776
0.6838046272493573
macro-F1: 0.6727176139435177
micro-F1: 0.659206510681587

进程已结束, 退出代码为 0
```

图 3: Linear 核

```
运行 SVM
C:\Users\19895\AppData\Local\Programs\Python\Python38\python.exe D:/干点正事/AI/LAB2_for_students/src1/SVM.py
train_num: 3554
test_num: 983
train_feature's shape: (3554, 8)
test_feature's shape: (983, 8)

  pccost    dcost      gap    pres    dres
0: -1.2883e+03 -8.7706e+03 5e+04 3e+00 3e-14
1: -8.4676e+02 -5.4394e+03 7e+03 2e-01 3e-14
2: -8.6401e+02 -1.5645e+03 8e+02 2e-02 2e-14
3: -9.3835e+02 -1.3874e+03 4e+02 8e-03 2e-14
4: -9.8443e+02 -1.1481e+03 2e+02 3e-03 2e-14
5: -9.8781e+02 -1.1399e+03 2e+02 3e-03 2e-14
6: -9.9845e+02 -1.1095e+03 1e+02 2e-03 2e-14
7: -1.0076e+03 -1.0845e+03 8e+01 9e-04 2e-14
8: -1.0143e+03 -1.0662e+03 5e+01 3e-04 2e-14
9: -1.0183e+03 -1.0561e+03 4e+01 2e-04 2e-14
10: -1.0229e+03 -1.0436e+03 2e+01 1e-14 3e-14
11: -1.0260e+03 -1.0374e+03 1e+01 4e-15 2e-14
12: -1.0270e+03 -1.0353e+03 8e+00 2e-14 2e-14
13: -1.0281e+03 -1.0332e+03 5e+00 1e-14 2e-14
14: -1.0289e+03 -1.0318e+03 3e+00 4e-14 2e-14
15: -1.0295e+03 -1.0308e+03 1e+00 5e-14 3e-14
16: -1.0297e+03 -1.0306e+03 9e-01 3e-14 2e-14
17: -1.0297e+03 -1.0306e+03 9e-01 1e-13 2e-14
18: -1.0299e+03 -1.0304e+03 5e-01 3e-14 2e-14
19: -1.0299e+03 -1.0303e+03 4e-01 7e-14 2e-14
20: -1.0300e+03 -1.0303e+03 3e-01 3e-14 2e-14
21: -1.0300e+03 -1.0302e+03 2e-01 1e-13 2e-14
22: -1.0301e+03 -1.0302e+03 9e-02 1e-13 2e-14
23: -1.0301e+03 -1.0301e+03 2e-02 4e-14 3e-14
24: -1.0301e+03 -1.0301e+03 4e-04 5e-14 3e-14
Optimal solution found.

  pccost    dcost      gap    pres    dres
0: -2.8738e+03 -9.7540e+03 4e+04 3e+00 5e-14
1: -2.8413e+03 -6.9428e+03 6e+03 2e-01 5e-14
2: -2.2639e+03 -3.0593e+03 9e+02 2e-02 5e-14
3: -2.4639e+03 -2.8447e+03 4e+02 9e-03 5e-14
4: -2.5666e+03 -2.7367e+03 2e+02 3e-03 5e-14
5: -2.6092e+03 -2.6870e+03 8e+01 1e-03 5e-14
6: -2.6244e+03 -2.6697e+03 5e+01 5e-04 5e-14
7: -2.6354e+03 -2.6570e+03 2e+01 2e-04 5e-14
8: -2.6421e+03 -2.6493e+03 7e+00 6e-05 5e-14
9: -2.6443e+03 -2.6469e+03 3e+00 1e-05 6e-14
10: -2.6453e+03 -2.6458e+03 6e-01 8e-07 6e-14
11: -2.6455e+03 -2.6456e+03 6e-02 7e-08 6e-14
12: -2.6455e+03 -2.6455e+03 1e-03 5e-10 6e-14
Optimal solution found.

  pccost    dcost      gap    pres    dres
0: -2.8738e+03 -9.7540e+03 4e+04 3e+00 5e-14
1: -2.8413e+03 -6.9428e+03 6e+03 2e-01 5e-14
2: -2.2639e+03 -3.0593e+03 9e+02 2e-02 5e-14
3: -2.4639e+03 -2.8447e+03 4e+02 9e-03 5e-14
4: -2.5666e+03 -2.7367e+03 2e+02 3e-03 5e-14
5: -2.6092e+03 -2.6870e+03 8e+01 1e-03 5e-14
6: -2.6244e+03 -2.6697e+03 5e+01 5e-04 5e-14
7: -2.6354e+03 -2.6570e+03 2e+01 2e-04 5e-14
8: -2.6421e+03 -2.6493e+03 7e+00 6e-05 5e-14
9: -2.6443e+03 -2.6469e+03 3e+00 1e-05 6e-14
10: -2.6453e+03 -2.6458e+03 6e-01 8e-07 6e-14
11: -2.6455e+03 -2.6456e+03 6e-02 7e-08 6e-14
12: -2.6455e+03 -2.6455e+03 1e-03 5e-10 6e-14
Optimal solution found.

  pccost    dcost      gap    pres    dres
0: -2.1684e+03 -9.2723e+03 4e+04 3e+00 6e-14
1: -1.4956e+03 -6.2132e+03 6e+03 1e-01 6e-14
2: -1.6217e+03 -2.4259e+03 9e+02 2e-02 5e-14
3: -1.7787e+03 -2.0679e+03 3e+02 5e-03 5e-14
4: -1.8099e+03 -2.0896e+03 2e+02 3e-03 5e-14
5: -1.8368e+03 -1.9541e+03 1e+02 1e-03 5e-14
6: -1.8566e+03 -1.9127e+03 6e+01 1e-04 6e-14
7: -1.8612e+03 -1.9039e+03 4e+01 1e-04 5e-14
8: -1.8682e+03 -1.8906e+03 2e+01 4e-05 5e-14
9: -1.8710e+03 -1.8858e+03 1e+01 2e-05 5e-14
10: -1.8737e+03 -1.8814e+03 8e+00 1e-05 5e-14
11: -1.8750e+03 -1.8794e+03 4e+00 4e-06 5e-14
12: -1.8757e+03 -1.8783e+03 3e+00 6e-07 6e-14
13: -1.8760e+03 -1.8779e+03 2e+00 3e-07 5e-14
14: -1.8763e+03 -1.8776e+03 1e+00 2e-07 5e-14
15: -1.8765e+03 -1.8774e+03 9e-01 1e-07 5e-14
16: -1.8767e+03 -1.8772e+03 5e-01 3e-08 5e-14
17: -1.8768e+03 -1.8771e+03 3e-01 1e-08 5e-14
18: -1.8769e+03 -1.8769e+03 1e-01 1e-09 6e-14
19: -1.8769e+03 -1.8769e+03 5e-02 2e-10 6e-14
20: -1.8769e+03 -1.8769e+03 7e-03 2e-11 6e-14
21: -1.8769e+03 -1.8769e+03 2e-04 1e-13 6e-14
Optimal solution found.
Acc: 0.582989460834181
0.6462195543175488
0.619136960803753
0.4695089242141775
macro-F1: 0.5782924797107806
micro-F1: 0.582989460834181

进程已结束，退出代码为 0
```

图 4: Gauss 核

```
运行: SVM
C:\Users\19895\AppData\Local\Programs\Python\Python38\python.exe D:/千点正事/AI/LAB2_for_students/src1/SVM.py
train_num: 3554
test_num: 983
train_feature's shape:(3554, 8)
test_feature's shape:(983, 8)
  pcost      dcost      gap      pres      dres
0: -1.3453e+03 -1.0113e+04  6e+04  3e+00  2e-12
1: -9.1009e+02 -6.9922e+03  1e+04  5e-01  2e-12
2: -8.3405e+02 -3.6676e+03  4e+03  2e-01  2e-12
3: -8.3788e+02 -2.2854e+03  2e+03  6e-02  2e-12
4: -9.0835e+02 -1.3485e+03  5e+02  2e-02  2e-12
5: -9.4480e+02 -1.2156e+03  3e+02  8e-03  2e-12
6: -9.6805e+02 -1.1397e+03  2e+02  4e-03  2e-12
7: -9.8196e+02 -1.0995e+03  1e+02  2e-03  2e-12
8: -9.9427e+02 -1.0662e+03  8e+01  1e-03  2e-12
9: -1.0006e+03 -1.0516e+03  5e+01  7e-04  2e-12
10: -1.0058e+03 -1.0395e+03  3e+01  4e-04  2e-12
11: -1.0091e+03 -1.0330e+03  2e+01  2e-04  2e-12
12: -1.0130e+03 -1.0253e+03  1e+01  8e-05  2e-12
13: -1.0145e+03 -1.0230e+03  9e+00  5e-05  2e-12
14: -1.0161e+03 -1.0203e+03  4e+00  1e-05  2e-12
15: -1.0170e+03 -1.0190e+03  2e+00  2e-06  2e-12
16: -1.0177e+03 -1.0183e+03  6e-01  3e-07  2e-12
17: -1.0179e+03 -1.0180e+03  8e-02  5e-08  2e-12
18: -1.0180e+03 -1.0180e+03  9e-03  4e-09  2e-12
Optimal solution found.
  pcost      dcost      gap      pres      dres
0: -2.9682e+03 -1.0603e+04  5e+04  3e+00  5e-12
1: -2.1011e+03 -8.1536e+03  1e+04  4e-01  4e-12
2: -2.1563e+03 -3.6288e+03  2e+03  3e-02  3e-12
3: -2.5255e+03 -2.8971e+03  4e+02  6e-03  4e-12
4: -2.5807e+03 -2.8329e+03  3e+02  3e-03  4e-12
5: -2.6559e+03 -2.7388e+03  9e+01  1e-03  4e-12
6: -2.6757e+03 -2.7154e+03  4e+01  4e-04  4e-12
7: -2.6882e+03 -2.7008e+03  1e+01  1e-04  4e-12
8: -2.6921e+03 -2.6963e+03  4e+00  3e-05  4e-12
9: -2.6934e+03 -2.6948e+03  1e+00  9e-06  4e-12
10: -2.6940e+03 -2.6941e+03  1e-01  1e-07  5e-12
11: -2.6941e+03 -2.6941e+03  1e-02  1e-08  5e-12
12: -2.6941e+03 -2.6941e+03  1e-03  1e-09  5e-12
Optimal solution found.
  pcost      dcost      gap      pres      dres
0: -2.9682e+03 -1.0603e+04  5e+04  3e+00  5e-12
1: -2.1011e+03 -8.1536e+03  1e+04  4e-01  4e-12
2: -2.1563e+03 -3.6288e+03  2e+03  3e-02  3e-12
3: -2.5255e+03 -2.8971e+03  4e+02  6e-03  4e-12
4: -2.5807e+03 -2.8329e+03  3e+02  3e-03  4e-12
5: -2.6559e+03 -2.7388e+03  9e+01  1e-03  4e-12
6: -2.6757e+03 -2.7154e+03  4e+01  4e-04  4e-12
7: -2.6882e+03 -2.7008e+03  1e+01  1e-04  4e-12
8: -2.6921e+03 -2.6963e+03  4e+00  3e-05  4e-12
9: -2.6934e+03 -2.6948e+03  1e+00  9e-06  4e-12
10: -2.6940e+03 -2.6941e+03  1e-01  1e-07  5e-12
11: -2.6941e+03 -2.6941e+03  1e-02  1e-08  5e-12
12: -2.6941e+03 -2.6941e+03  1e-03  1e-09  5e-12
Optimal solution found.
  pcost      dcost      gap      pres      dres
0: -2.2173e+03 -1.1255e+04  6e+04  4e+00  4e-12
1: -1.4797e+03 -8.5162e+03  1e+04  4e-01  4e-12
2: -1.4684e+03 -3.1823e+03  2e+03  3e-02  3e-12
3: -1.6725e+03 -2.2831e+03  6e+02  1e-02  3e-12
4: -1.7472e+03 -2.1379e+03  4e+02  4e-03  3e-12
5: -1.7940e+03 -2.0200e+03  2e+02  2e-03  4e-12
6: -1.8084e+03 -1.9878e+03  2e+02  1e-03  3e-12
7: -1.8282e+03 -1.9451e+03  1e+02  7e-04  3e-12
8: -1.8350e+03 -1.9288e+03  9e+01  5e-04  3e-12
9: -1.8393e+03 -1.9214e+03  8e+01  4e-04  3e-12
10: -1.8486e+03 -1.8987e+03  5e+01  1e-04  4e-12
11: -1.8545e+03 -1.8887e+03  3e+01  5e-05  4e-12
12: -1.8580e+03 -1.8831e+03  3e+01  3e-05  4e-12
13: -1.8601e+03 -1.8795e+03  2e+01  2e-05  4e-12
14: -1.8629e+03 -1.8754e+03  1e+01  9e-06  4e-12
15: -1.8642e+03 -1.8736e+03  9e+00  6e-06  3e-12
16: -1.8667e+03 -1.8702e+03  4e+00  1e-06  4e-12
17: -1.8679e+03 -1.8689e+03  1e+00  1e-07  4e-12
18: -1.8683e+03 -1.8684e+03  8e-02  7e-09  4e-12
19: -1.8683e+03 -1.8683e+03  1e-02  9e-10  4e-12
20: -1.8683e+03 -1.8683e+03  4e-04  2e-11  4e-12
Optimal solution found.
Acc: 0.5778229908443541
0.5049180327868853
0.6070409134157946
0.5639344262295083
macro-F1: 0.5586311241440627
micro-F1: 0.5778229908443541

进程已结束,退出代码为 0
```

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%

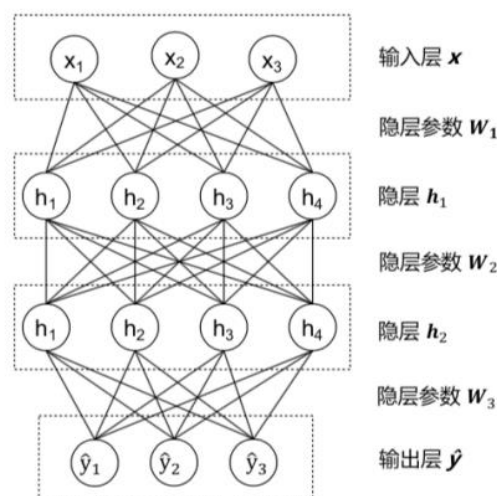
通过比较发现，三种核预测的准确率差别并不大，对于具体问题不同的核的表现也不尽相同，但是在计算速度上线性核的速度明显快于其他核，在实际问题中如果线性核的效果较好，则直接使用线性核足矣。

3. 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，即输入的特征维为 5，输出的类别个数为 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。



$$\mathbf{h}_1 = \mathbf{s}_1(\mathbf{W}_1 \mathbf{x})$$

$$\mathbf{h}_2 = \mathbf{s}_2(\mathbf{W}_2 \mathbf{h}_1)$$

$$\hat{\mathbf{y}} = \mathbf{s}_3(\mathbf{W}_3 \mathbf{h}_2)$$

$$L = l(\hat{\mathbf{y}}, \mathbf{y})$$

$$\mathbf{s}_1 = \mathbf{s}_2 = \sigma$$

$$\mathbf{s}_3 = \text{softmax}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3) = \frac{1}{e^{x_1} + e^{x_2} + e^{x_3}} (e^{x_1}, e^{x_2}, e^{x_3})$$

$$l(\hat{\mathbf{y}}, \mathbf{y}) = \text{CrossEntropy}(\mathbf{y}, \hat{\mathbf{y}}) = -\log \hat{y}_i, i = y$$

$$(l' \mathbf{s}'_3)_i = \begin{cases} \hat{y}_i - 1, i = y \\ \hat{y}_i, i \neq y \end{cases}$$

$$\frac{\partial L}{\partial \mathbf{W}_1} = (\mathbf{W}_2^T (\mathbf{W}_3^T (l' \mathbf{s}'_3) \odot \mathbf{s}'_2) \odot \mathbf{s}'_1) \mathbf{x}^T$$

$$\frac{\partial L}{\partial \mathbf{W}_2} = (\mathbf{W}_3^T (l' \mathbf{s}'_3) \odot \mathbf{s}'_2) \mathbf{h}_1^T$$

$$\frac{\partial L}{\partial \mathbf{W}_3} = (l' \mathbf{s}'_3) \mathbf{h}_2^T$$

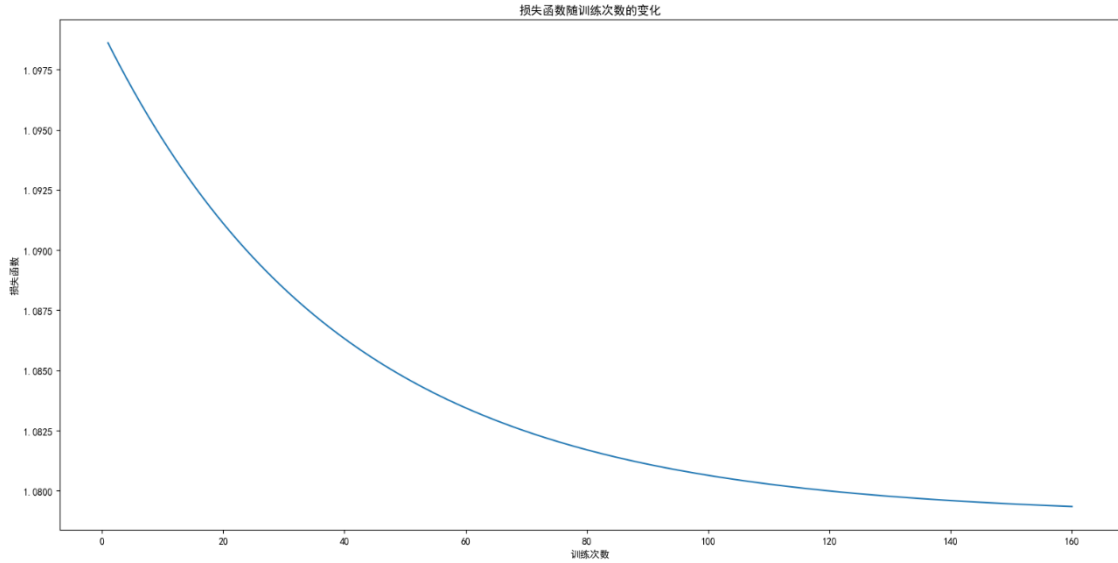
我算法的大致步骤是：

- (1) 初始化 $\mathbf{W}_1, \mathbf{W}_2, \mathbf{W}_3$
- (2) 前向传播得到预测结果 $\hat{\mathbf{y}}$
- (3) 依据矩阵求导公式计算 $\mathbf{W}_1, \mathbf{W}_2, \mathbf{W}_3$ 的梯度，更新矩阵
- (4) 若 $|l^{(k+1)}(\hat{\mathbf{y}}, \mathbf{y}) - l^{(k)}(\hat{\mathbf{y}}, \mathbf{y})| < \varepsilon$ ，则算法终止，否则回到(2)

实验结果如下：

The screenshot displays a Jupyter Notebook environment with a file explorer on the left showing 'MLP_manual.py'. The main area contains a Python script for training a neural network. The script defines a simple neural network architecture with three layers (myd1, myd2, myd3) and a loss function (myloss). It then trains the model using torch.nn optimizers and prints the loss and accuracy at each step. The output shows the loss decreasing from approximately 1.09 to 0.005 over 112 iterations.

```
C:\ProgramData\Anaconda3\python.exe D:/千点正/AI/LAB2_for_students/src2/MLP_manual.py
myd1: [[0.0, 0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0, 0.0]]
myd2: [[-1.1288865623579185e-05, -1.1288865623579185e-05, -1.1288865623579185e-05, -1.1288865623579185e-05], [-1.1288865623579185e-05, -1.1288865623579185e-05, -1.1288865623579185e-05, -1.1288865623579185e-05], [-1.1288865623579185e-05, -1.1288865623579185e-05, -1.1288865623579185e-05, -1.1288865623579185e-05]]
myd3: [[-0.012731784487236957, -0.012731784487236957, -0.012731784487236957, -0.012731784487236957], [0.0858414997312398256, 0.0858414997312398256, 0.0858414997312398256, 0.0858414997312398256]]
myloss: 1.091839803426615
训练了 112 次收敛
torch d1: [[0.0, 0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0, 0.0]]
torch d2: [[-1.1288865623574888e-05, -1.1288865623574888e-05, -1.1288865623574888e-05, -1.1288865623574888e-05], [-1.1288865623574888e-05, -1.1288865623574888e-05, -1.1288865623574888e-05, -1.1288865623574888e-05], [-1.1288865623574888e-05, -1.1288865623574888e-05, -1.1288865623574888e-05, -1.1288865623574888e-05]]
torch d3: [[-0.012731784487236923, -0.012731784487236923, -0.012731784487236923, -0.012731784487236923], [0.0858414997312398976, 0.0858414997312398976, 0.0858414997312398976, 0.0858414997312398976]]
torch_loss: 1.0918398034266163
训练了 112 次收敛
进程已结束,退出代码为 0
```



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）

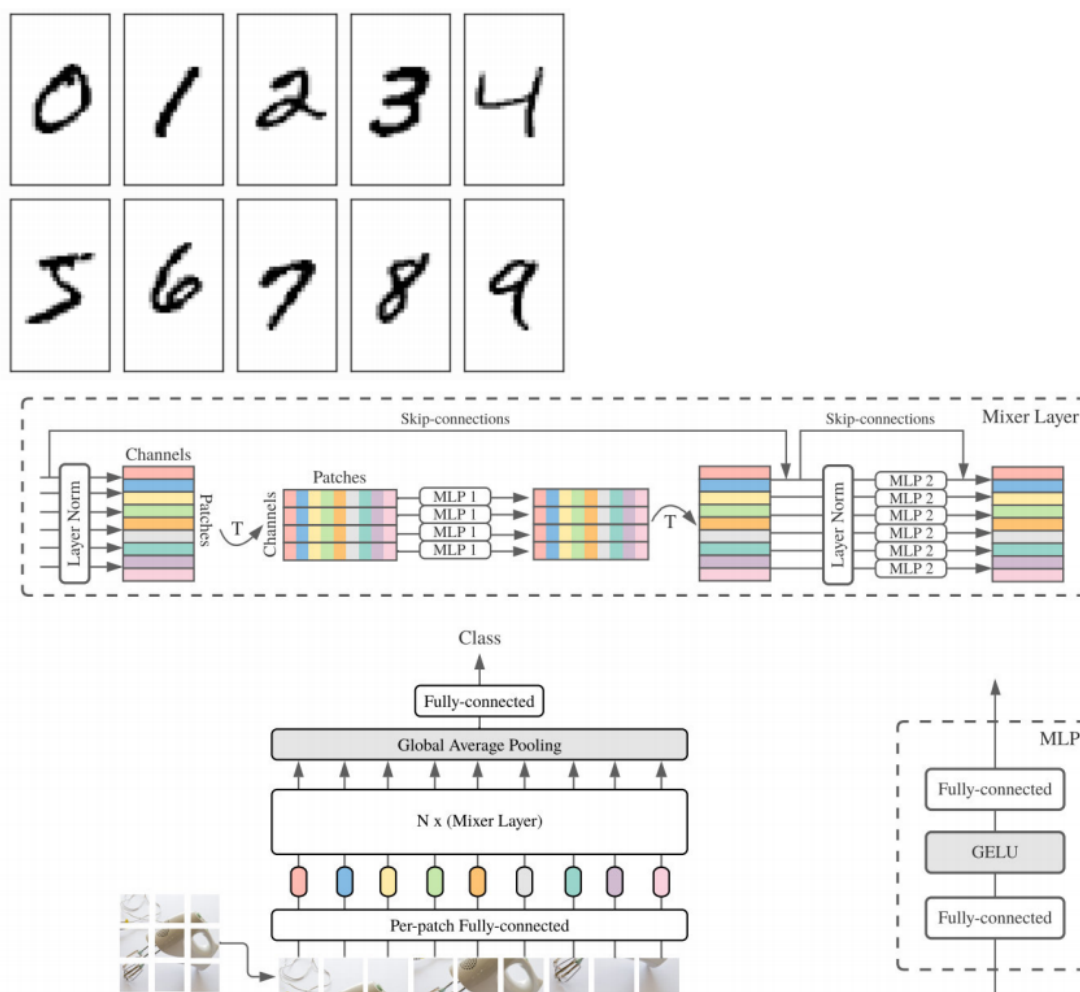


Figure 1: MLP-Mixer consists of per-patch linear embeddings, Mixer layers, and a classifier head. Mixer layers contain one token-mixing MLP and one channel-mixing MLP, each consisting of two fully-connected layers and a GELU nonlinearity. Other components include: skip-connections, dropout, layer norm on the channels, and linear classifier head.

<https://blog.csdn.net/d01348881/>

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 $S \times S$ 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 $S \times C$ 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 $C \times S$ 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 $S \times C$ 后与原始输入 x 进行跨层直连得到 x_2
- (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 $S \times C$ 的输出与 x_2 跨层直连得到 $S \times C$ 的最终输出。

输出层：

单层全连接，输入是 $S \times C$ 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：

```
MLP_Mixer x
C:\Users\19895\AppData\Local\conda\conda\envs\env_pytorch\python.exe D:/
C:\Users\19895\AppData\Local\conda\conda\envs\env_pytorch\lib\site-packa
warnings.warn(warning.format(ret))
Train Epoch: 0/5 [0/60000] Loss: 2.379466
Train Epoch: 0/5 [12800/60000] Loss: 0.810180
Train Epoch: 0/5 [25600/60000] Loss: 0.378310
Train Epoch: 0/5 [38400/60000] Loss: 0.260573
Train Epoch: 0/5 [51200/60000] Loss: 0.187051
Train Epoch: 1/5 [0/60000] Loss: 0.212707
Train Epoch: 1/5 [12800/60000] Loss: 0.160789
Train Epoch: 1/5 [25600/60000] Loss: 0.234418
Train Epoch: 1/5 [38400/60000] Loss: 0.111569
Train Epoch: 1/5 [51200/60000] Loss: 0.169957
Train Epoch: 2/5 [0/60000] Loss: 0.145229
Train Epoch: 2/5 [12800/60000] Loss: 0.113614
Train Epoch: 2/5 [25600/60000] Loss: 0.113110
Train Epoch: 2/5 [38400/60000] Loss: 0.091772
Train Epoch: 2/5 [51200/60000] Loss: 0.249947
Train Epoch: 3/5 [0/60000] Loss: 0.043583
Train Epoch: 3/5 [12800/60000] Loss: 0.055656
Train Epoch: 3/5 [25600/60000] Loss: 0.096923
Train Epoch: 3/5 [38400/60000] Loss: 0.146514
Train Epoch: 3/5 [51200/60000] Loss: 0.090126
Train Epoch: 4/5 [0/60000] Loss: 0.086184
Train Epoch: 4/5 [12800/60000] Loss: 0.062057
Train Epoch: 4/5 [25600/60000] Loss: 0.054231
Train Epoch: 4/5 [38400/60000] Loss: 0.026461
Train Epoch: 4/5 [51200/60000] Loss: 0.018258
Test set: Average loss: 0.0980 Acc 0.97

进程已结束，退出代码为 0
```

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！