

AI实验一

袁雨 PB20151804

一、实验要求

基本实验流程：实现一个模型，然后在 **训练集** 上训练（根据训练样本的误差优化参数），每轮迭代后在 **验证集** 上验证，根据验证样本的误差决定是否停止训练、是否需要修改模型结构等超参数，如果需要修改超参数则修改模型后重新开始训练。最终获得一个最优超参数且完成训练的模型，用于预测 **测试集** 的结果。

具体要求：

1. 遵照助教划分的训练集、验证集、测试集，只在训练集上训练。
2. 不限制编程语言。
3. 使用多层感知机（全连接网络），分别测试只包含2层、3层、4层神经元的模型（即3个模型），自行调整其他超参数/参数（例如每层神经元数量、使用什么激活函数、是否使用 dropout 或 bias、epoch 数等属于超参数）；
4. 使用 SVM 模型，自行调整其他超参数/参数；
5. 根据**验证集**的 **F1 score** 选择上述4个模型的超参数，**重点**记录选择过程和选择理由；
6. 分析最终4个模型在验证集上的拟合情况，例如不同模型的过拟合、欠拟合程度等；
7. 分析4个模型在**测试集**的 F1 score、ROC 曲线及 AUC（比如是否符合验证集的预期，是否观察到其他现象）；

二、准备工作


1. 实验环境




①机器设备情况：

设备规格

MateBook X Pro

设备名称	LAPTOP-6UPMBQBV
处理器	Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz 2.11 GHz
机带 RAM	16.0 GB (15.8 GB 可用)
设备 ID	07E1508B-4DF0-4473-AABB-8D9550F8533B
产品 ID	00342-35667-73251-AAOEM
系统类型	64 位操作系统, 基于 x64 的处理器
笔和触控	没有可用于此显示器的笔或触控输入

▼  显示适配器

-  Intel(R) UHD Graphics
-  NVIDIA GeForce MX250
-  Virtual Display Device

NVIDIA-SMI 512.15				Driver Version: 512.15		CUDA Version: 11.6	
GPU	Name	TCC/WDDM		Bus-Id	Disp. A	Volatile	Uncorr. ECC
Fan	Temp	Perf	Pwr:Usage/Cap	Memory-Usage		GPU-Util	Compute M.
						MIG M.	
0	NVIDIA GeForce ...	WDDM		00000000:01:00.0	Off	N/A	
N/A	60C	P0	N/A / N/A	0MiB /	2048MiB	3%	Default
						N/A	
Processes:							
GPU	GI	CI	PID	Type	Process name	GPU Memory	
	ID	ID				Usage	
No running processes found							

②所用语言: python 3.10

③所用库:

```
import torch
from torch.utils.data import Dataset # 抽象类 继承
from torch.utils.data import DataLoader # 可实例化
import numpy as np
import pandas as pd
import torch.nn.functional as f # 激励函数的库
import matplotlib.pyplot as plt
from sklearn import metrics
from sklearn import svm
from sklearn.metrics import f1_score
import gc
from sklearn.metrics import roc_auc_score as auc
from sklearn.metrics import roc_curve
```

2. 对实验数据的预处理过程和读取方式

①预处理

对于感知机, 不进行数据预处理; 对于SVM模型, 机器性能不够, 选取前10000个样本进行实验。

②读取方式

导入pandas包并调用read_csv函数, 从csv文件中加载原始数据集, 分别装载数据和标签。实现一个dataset对象后, 将其传入到dataloader中, 利用多进程来加速batch data的处理。并将每次取出的训练集和验证集的batch数据放到GPU上。以训练集为例:

```
data = pd.read_csv(self.root) # 从CSV文件中加载原始数据集
self.label = data.iloc[:, -1] # 装载标签
self.data = np.array(data.iloc[:, 1:-1]) # 装载数据
train_dataset = GetData('train')
# 创建加载器
```

```

train_loader = DataLoader(dataset=train_dataset, batch_size=100, shuffle=
True, num_workers=0)    # 并行进程数
# device
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
for i, (data, target) in enumerate(train_loader):    # enumerate得到循环次
数
    data = data.to(device)

```

三、调整超参数

1. 神经元数量

①输入层：个数等于特征数；

②隐藏层：可能不只一层，神经元数目也不确定，需要根据实验和经验选择；

③输出层：个数为输出的种类。

若隐层节点数太少，网络可能根本不能训练或网络性能很差；

若隐层节点数太多，虽然可使网络的系统误差减小，但一方面使网络训练时间延长，另一方面，训练容易陷入局部极小点而得不到最优点，也是训练时出现“过拟合”的内在原因。因此，合理隐层节点数应在综合考虑网络结构复杂程度和误差大小的情况下确定。

二层感知机	epoch=5				
hidden1_num	验证集F1 score				
256	0.734879				
512	0.735318				
1024	0.735523				
1536	0.735574				
2048	0.735648				
4096	0.735568				
三层感知机	epoch=10				
hidden1_num	验证集F1 score	hidden2_num	验证集F1 score		
128	0.734459	128	0.735401		
256	0.735188	512	0.735512		
512	0.735168	1024	0.735621		
1024	0.735351	1536	0.735565		
1536	0.735401				
2048	0.735360				
四层感知机	epoch=5				
hidden1_num	验证集F1 score	hidden2_num	验证集F1 score	hidden3_num	验证集F1 score
256	0.735198	128	0.735198	256	0.736681
512	0.736228	512	0.736143	512	0.735417
1024	0.735544	1024	0.736362	1024	0.737910
		1536	0.736681	1536	0.736654
		2048	0.736366		

结合实验，使用控制变量法（下同），当第一个隐含层的神经元数目增加时，验证集上的 F1 score先上升，继续增加神经元数目时，发现训练集准确率上升，但验证集F1 score 下降，即产生了过拟合。

因此，对于二层感知机，选择2048较为合适，对于三层感知机，选择1536与1024较为合适，对于四层感知机，选择512、1536和1024较为合适。

2. 学习率

学习率影响系统学习过程的稳定性。

大的学习率，在算法优化的前期会加速学习，使得模型更容易接近局部或全局最优解。但是在后期会有较大波动，可能使网络权值每一次的修正量过大，甚至会导致权值在修正过程中超出某个误差的极小值呈不规则跳跃而不收敛；

但过小的学习率导致学习时间过长，不过能保证收敛于某个极小值。所以，一般倾向选取较小的学习率以保证学习过程的收敛性（稳定性），通常在0.01~0.8之间。

	二层感知机	epoch=5	
	学习率	验证集F1 score	
	0.001	0.735648	
	0.01	0.736409	
	0.1	0.746867	
	0.3	0.748359	
	0.5	0.741105	
	0.6	0.749863	
	0.7	0.745953	
	0.8	0.744027	
	1	0.737179	
	三层感知机	epoch=10	
	学习率	验证集F1 score	
	0.001	0.735621	
	0.01	0.736842	
	0.05	0.740068	
	0.1	0.750630	
	0.2	0.747523	
	0.3	0.733105	
	1	0.733599	
	四层感知机	epoch=5	
	学习率	验证集F1 score	
	0.01	0.737910	
	0.05	0.736240	
	0.1	0.745100	
	0.2	0.743683	
	0.3	0.736296	

结合实验，对于二层感知机，选择0.1；对于三层感知机，选择0.6；对于四层感知机，选择0.1。

3. dropout

Dropout 以概率 p 来丢弃神经元，并且让别的神经元以概率 $q = 1 - p$ ，进行保留。每一个神经元都有相同的概率被丢弃和保留。

取平均的作用：先回到正常的模型（没有dropout），我们用相同的训练数据去训练不同的神经网络，一般会得到不同的结果，此时我们可以采用“所有结果取均值”或者“多数取胜的投票策略”去决定最终结果。这种“综合起来取平均”的策略通常可以有效防止过拟合问题。因为不同的网络可能产生不同的过拟合，取平均则有可能让一些“相反的”拟合互相抵消。dropout掉不同的隐藏神经元就类似在训练不同的网络（随机删掉一半隐藏神经元导致网络结构已经不同），整个dropout过程就相当于对很多个不同的神经网络取平均。而不同的网络产生不同的过拟合，一些互为“反向”的拟合相互抵消就可以达到整体上减少过拟合。

减少神经元之间复杂的共适应关系：因为dropout程序导致两个神经元不一定每次都在一个dropout网络中出现。（这样权值的更新不再依赖于有固定关系的隐含节点的共同作用，阻止了某些特征仅仅在其它特定特征下才有效果的情况）。迫使网络去学习更加鲁棒的特征（这些特征在其它的神经元的随机子集中也存在）。换句话说假如我们的神经网络是在做出某种预测，它不应该对一些特定的线索片段太过敏感，即使丢失特定的线索，它也应该可以从众多其它线索中学习一些共同的模式（鲁棒性）。

三层感知机	epoch=10
dropout	验证集F1 score
0.5	0.750630
0.3	0.743781
0.1	0.747706

结合实验，以三层感知机为例，经过交叉验证，可见当dropout率取0.5时，F1 score最高。p=0.5时，随机性更大，dropout随机生成的网络结构最多。

4. 损失函数

损失函数（loss function）是用来估量你模型的预测值 $f(x)$ 与真实值 Y 的不一致程度，它是一个非负实值函数，通常使用 $L(Y, f(x))$ 来表示，损失函数越小，模型的鲁棒性就越好。损失函数是经验风险函数的核心部分，也是结构风险函数重要组成部分。模型的结构风险函数包括了经验风险项和正则项，通常可以表示成如下式子：

$$\theta^* = \arg \min_{\theta} \frac{1}{N} \sum_{i=1}^N L(y_i, f(x_i; \theta)) + \lambda \Phi(\theta)$$

其中，前面的均值函数表示的是经验风险函数， L 代表的是损失函数，后面的 Φ 是正则化项（regularizer）或者叫惩罚项（penalty term），它可以是 $L1$ ，也可以是 $L2$ ，或者其他正则函数。整个式子表示的意思是找到使目标函数最小时的 θ 值。下面主要列出几种常见的损失函数。

5. 激活函数

神经网络中的每个神经元节点接受上一层神经元的输出值作为本神经元的输入值，并将输入值传递给下一层，输入层神经元节点会将输入属性值直接传递给下一层（隐层或输出层）。在多层神经网络中，上层节点的输出和下层节点的输入之间具有一个函数关系，这个函数称为激活函数。

如果不用激活函数（其实相当于激活函数是 $f(x) = x$ ），在这种情况下你每一层节点的输入都是上层输出的线性函数，很容易验证，无论你神经网络有多少层，输出都是输入的线性组合，与没有隐藏层效果相当，这种情况就是最原始的感知机了，那么网络的逼近能力就相当有限。正因为上面

的原因，我们决定引入非线性函数作为激活函数，这样深层神经网络表达能力就更加强大（不再是输入的线性组合，而是几乎可以逼近任意函数）。

1. 均方差损失函数+Sigmoid激活函数：梯度变化值很小，导致我们的W,b更新到极值的速度较慢，也就是我们的算法收敛速度较慢；
2. 使用交叉熵损失函数+Sigmoid激活函数改进DNN算法收敛速度；
3. 使用对数似然损失函数和softmax激活函数进行DNN分类输出；
4. 使用ReLU激活函数部分解决梯度消失问题的办法，尤其是在CNN模型中。

损失函数	激活函数	验证集F1 score
CrossEntropy	relu	0.752009
CrossEntropy	sigmoid	0.735686

结合实验，选择交叉熵损失函数+ReLU激活函数。

6. epoch

当一个完整的数据集通过了神经网络一次并且返回了一次，这个过程称为一次>epoch。（也就是说，所有训练样本在神经网络中都进行了一次正向传播和一次反向传播）再通俗一点，一个Epoch就是将所有训练样本训练一次的过程。在神经网络中传递完整的数据集一次是不够的，而且我们需要将完整的数据集在同样的神经网络中传递多次。但我们使用的是有限的数据集，并且我们使用一个迭代过程即梯度下降来优化学习过程。因此仅仅更新一次或者说使用一个epoch是不够的。随着epoch数量增加，神经网络中的权重的更新次数也在增加，曲线从欠拟合变得过拟合。

对于不同的数据集，合适的epoch数目是不一样的。且数据的多样性会影响合适的epoch的数量。

三层感知机	
epoch数	验证集F1 score
5	0.746976
10	0.750228
20	0.747059

结合实验，发现随着epoch数的增加，训练集的准确率上升，验证机的准确率和F1 score先上升，后产生过拟合，开始下降。综合选择epoch=10。

4. SVM

(1) 软间隔

硬间隔可以将所有样本点划分正确且都在间隔边界之外，即所有样本点都满足 $y_i(w^T x_i + b) \geq 1$ 。但硬间隔有两个缺点：1. 不适用于线性不可分数据集。2. 对离群点（outlier）敏感。

为了缓解这些问题，引入了“软间隔”，即允许一些样本点跨越间隔边界甚至是超平面。于是为每个样本点引入松弛变量 ξ_i ，优化问题变为：

$$\begin{aligned} \min_{w, b, \xi} \quad & \frac{1}{2} \|w\|^2 + C \sum_{i=1}^m \xi_i \\ \text{s.t.} \quad & y_i (w^T x_i + b) \geq 1 - \xi_i, \quad i = 1, 2, \dots, m \\ & \xi_i \geq 0, \quad i = 1, 2, \dots, m \end{aligned}$$

由上式可以看出：

- ①离群点的松弛变量值越大，点就离间隔边界越远。
 - ②所有没离群的点松弛变量都等于0，即这些点都满足 $y_i(wTx_i+b) \geq 1$ 。
 - ③ $C>0$ 被称为惩罚参数，即为 scikit-learn 中的 svm 超参数C。
- 当C设的越大，意味着对离群点的惩罚就越大，最终就会有较少的点跨过间隔边界，模型也会变得复杂，训练时间也会更长，容易过拟合。而C设的越小，则较多的点会跨过间隔边界，最终形成的模型较为平滑，准确度不高，容易欠拟合。默认情况下C为1，通常来说这都是一个合理的参数。 如果我们的数据很嘈杂，那我们往往减小C。

(2) 核函数

硬间隔和软间隔 svm 主要是用于处理线性分类问题，然而现实中很多分类问题是非线性的。所以在此引入核函数（kernel function），构造非线性svm。核函数的主要作用是将样本从原始空间映射到一个更高维的特征空间，使得样本在这个特征空间内线性可分，但回到原来的二维空间中，决策边界就变成非线性的了。

核函数	用处	公式
linear kernel	线性可分时，特征数量多时，样本数量多再补充一些特征时，linear kernel可以是RBF kernel的特殊情况	
Polynomial kernel	image processing，参数比RBF多，取值范围是(0,inf)	
Gaussian radial basis function (RBF)	通用，线性不可分时，特征维数少 样本数量正常时，在没有先验知识时用，取值在[0,1]	
Sigmoid kernel	生成神经网络，在某些参数下和RBF很像，可能在某些参数下是无效的	
Gaussian kernel	通用，在没有先验知识时用	
Laplace RBF kernel	通用，在没有先验知识时用	
Hyperbolic tangent kernel	neural networks中用	
Bessel function of the first kind Kernel	可消除函数中的交叉项	
ANOVA radial basis kernel	回归问题	
Linear splines kernel in one-dimension	text categorization，回归问题，处理大型稀疏向量	

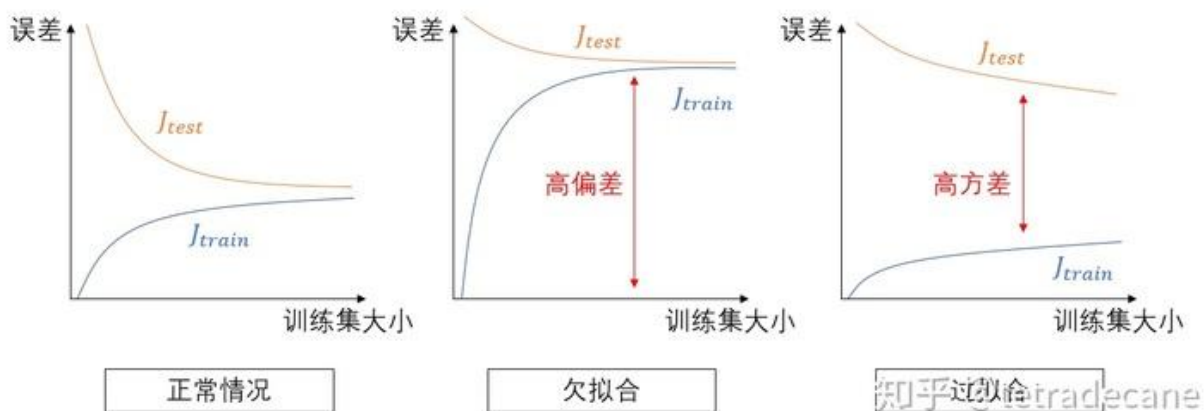
SVM 1000个样本		
核函数	软间隔 (C=?)	验证集F1 score
linear	0.01	0.735582
linear	0.1	0.735575
linear	0.25	0.734034
linear	0.5	0.721485
linear	0.75	0.712738
linear	1	0.705512
sigmoid	0.25	0.492576
sigmoid	0.5	0.735575
sigmoid	0.6	0.735582
sigmoid	0.65	0.735582
sigmoid	0.75	0.735582
sigmoid	1	0.735574
sigmoid	1.5	0.735541
rbf	0.1	0.521188
rbf	0.5	0.735582
rbf	0.6	0.735582
rbf	0.75	0.735582
rbf	0.8	0.735582
rbf	1	0.735582
rbf	1.5	0.735555
poly	0.01	0.054651
poly	0.1	0.735582
poly	0.15	0.735582
poly	0.25	0.735563
poly	0.5	0.731017
poly	1	0.706200

结合实验，选择 $C=0.75$ ， $\text{kernel}='rbf'$ 。

四、测试结果

1. 4个模型在验证集上的结果

拟合情况：



(1) 感知机

二层感知机：

Epoch: 1 Training Loss: 0.553440

Accuracy of the network on the train_data: 75.674000 %
Epoch: 2 Training Loss: 0.546984
Accuracy of the network on the train_data: 75.952143 %
Epoch: 3 Training Loss: 0.544605
Accuracy of the network on the train_data: 75.584000 %
Epoch: 4 Training Loss: 0.543213
Accuracy of the network on the train_data: 75.812714 %
Epoch: 5 Training Loss: 0.542315
Accuracy of the network on the train_data: 75.998286 %
Epoch: 6 Training Loss: 0.541794
Accuracy of the network on the train_data: 76.166714 %
Epoch: 7 Training Loss: 0.541198
Accuracy of the network on the train_data: 75.577143 %
Epoch: 8 Training Loss: 0.540784
Accuracy of the network on the train_data: 76.262000 %
Epoch: 9 Training Loss: 0.540348
Accuracy of the network on the train_data: 76.281000 %
Epoch: 10 Training Loss: 0.540003
Accuracy of the network on the train_data: 76.083286 %
验证集:
Accuracy of the network on the valid_data: 75.536500 %
F1 Score:0.751504

三层感知机:

Epoch: 1 Training Loss: 0.556749
Accuracy of the network on the train_data: 75.811714 %
Epoch: 2 Training Loss: 0.546681
Accuracy of the network on the train_data: 75.903714 %
Epoch: 3 Training Loss: 0.543411
Accuracy of the network on the train_data: 75.977571 %
Epoch: 4 Training Loss: 0.541265
Accuracy of the network on the train_data: 76.115143 %
Epoch: 5 Training Loss: 0.539739
Accuracy of the network on the train_data: 76.119143 %
Epoch: 6 Training Loss: 0.538828
Accuracy of the network on the train_data: 76.064429 %
Epoch: 7 Training Loss: 0.538315
Accuracy of the network on the train_data: 76.182143 %
Epoch: 8 Training Loss: 0.537963
Accuracy of the network on the train_data: 76.234429 %
Epoch: 9 Training Loss: 0.537466
Accuracy of the network on the train_data: 76.278571 %
Epoch: 10 Training Loss: 0.537259
Accuracy of the network on the train_data: 76.179857 %
验证集:
Accuracy of the network on the test_data: 75.745000 %
F1 Score:0.750228

四层感知机:

Epoch: 1 Training Loss: 0.562019
Accuracy of the network on the train_data: 75.049714 %

Epoch: 2 Training Loss: 0.548549
 Accuracy of the network on the train_data: 75.868429 %
 Epoch: 3 Training Loss: 0.544525
 Accuracy of the network on the train_data: 75.911857 %
 Epoch: 4 Training Loss: 0.542005
 Accuracy of the network on the train_data: 76.006429 %
 Epoch: 5 Training Loss: 0.540476
 Accuracy of the network on the train_data: 76.106714 %
 Epoch: 6 Training Loss: 0.539489
 Accuracy of the network on the train_data: 76.137714 %
 Epoch: 7 Training Loss: 0.538826
 Accuracy of the network on the train_data: 76.113143 %
 Epoch: 8 Training Loss: 0.538455
 Accuracy of the network on the train_data: 76.155000 %
 Epoch: 9 Training Loss: 0.538096
 Accuracy of the network on the train_data: 76.216286 %
 Epoch: 10 Training Loss: 0.537884
 Accuracy of the network on the train_data: 76.205571 %
 验证集:
 Accuracy of the network on the test_data: 75.771000 %
 F1 Score:0.743542

二层感知机		
样本量	训练集准确率	验证集准确率
1000	77.300000%	73.209500%
10000	76.030000%	74.580500%
100000	75.714000%	75.151500%
700000	76.083286%	75.536500%
三层感知机		
样本量	训练集准确率	验证集准确率
10000	75.610000%	75.680000%
100000	75.996000%	75.387000%
700000	76.179857%	75.745000%
四层感知机		
样本量	训练集准确率	验证集准确率
10000	75.510000%	75.222000%
100000	75.699000%	75.124000%
700000	76.205571%	75.771000%

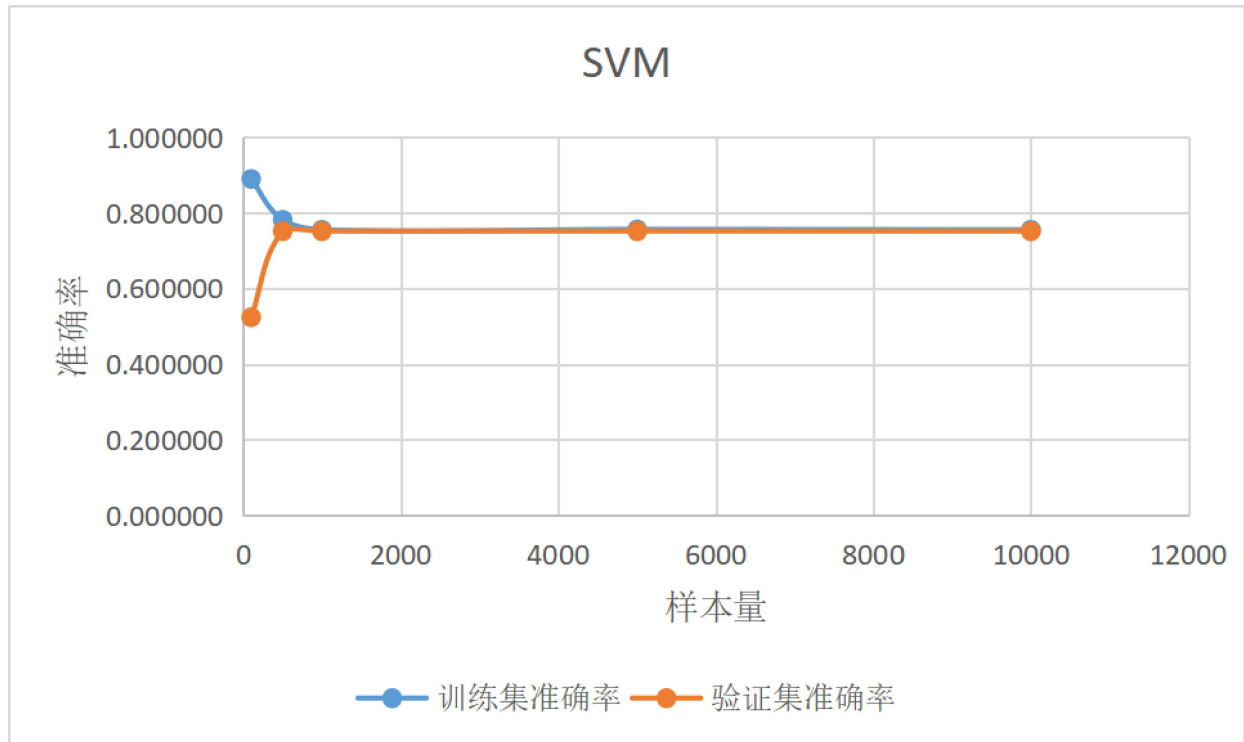
训练集和验证集的准确率均趋于定值，且两个定值接近，没有出现欠拟合或过拟合。

(2) SVM

训练集:
 准确率: 0.755700
 验证集:
 准确率: 0.752190

F1 score = 0.735582

SVM		
样本量	训练集准确率	验证集准确率
100	0.890000	0.523720
500	0.782000	0.752190
1000	0.755700	0.752190
5000	0.756600	0.752190
10000	0.755700	0.752190



随着样本量的增大，训练集准确率先下降后趋于定值，验证集准确率先上升后趋于定值，且两个定值接近，准确率高。故属于正常情况，没有出现欠拟合或者过拟合。

2. 4个模型在测试集上的结果

(1) 二层感知机

验证集：

Accuracy of the network on the valid_data: 75.536500 %

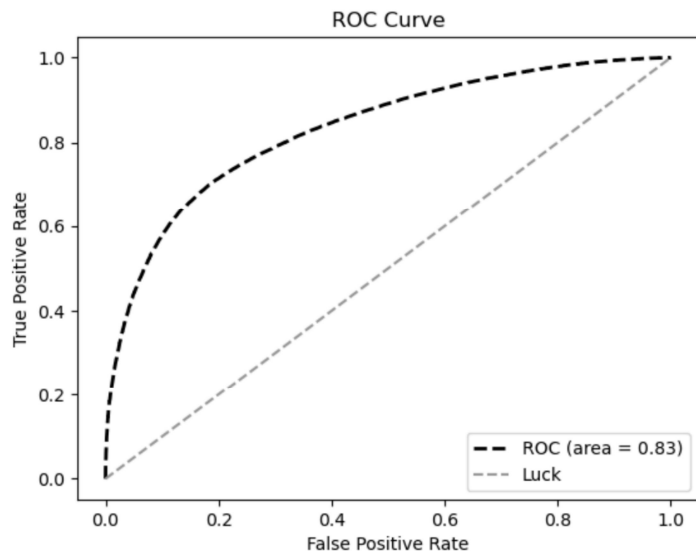
F1 Score:0.751504

测试集：

Accuracy of the network on the test_data: 75.465000 %

F1 Score:0.752750

AUC:0.831638



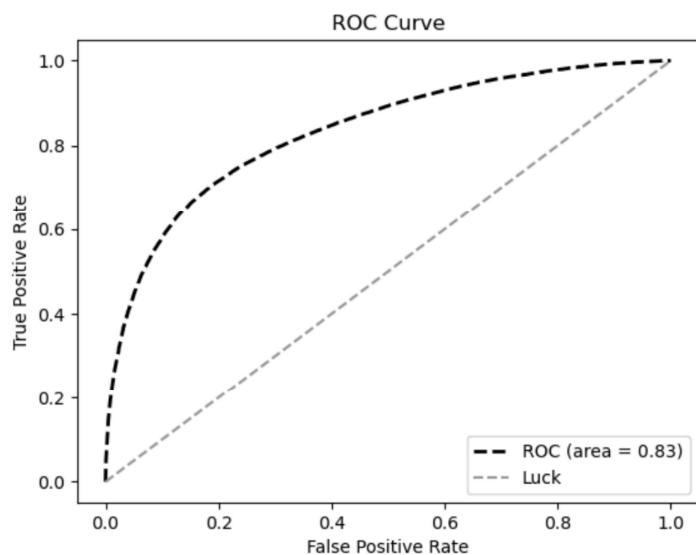
(2) 三层感知机

验证集：

Accuracy of the network on the test_data: 75.745000 %
F1 Score:0.750228

测试集：

Accuracy of the network on the test_data: 75.763000 %
F1 Score:0.751300
AUC:0.833659



(3) 四层感知机

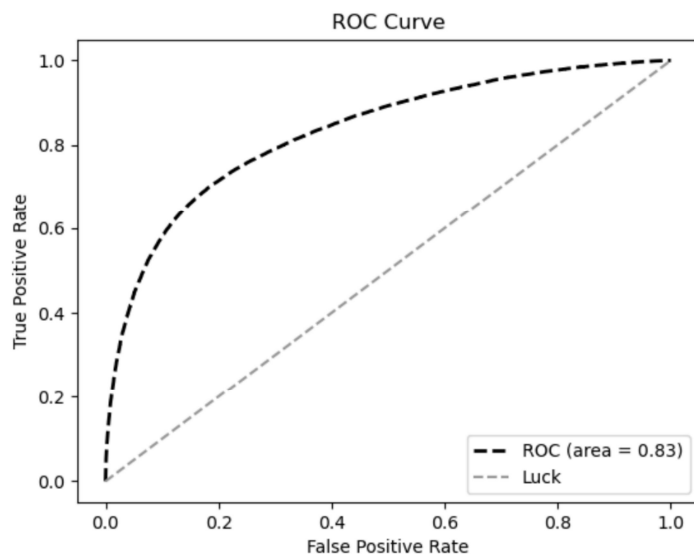
验证集：

Accuracy of the network on the test_data: 75.771000 %
F1 Score:0.743542

测试集：

Accuracy of the network on the test_data: 75.721000 %
F1 Score:0.745158

AUC:0.831373



(4) SVM

验证集:

准确率: 0.752190

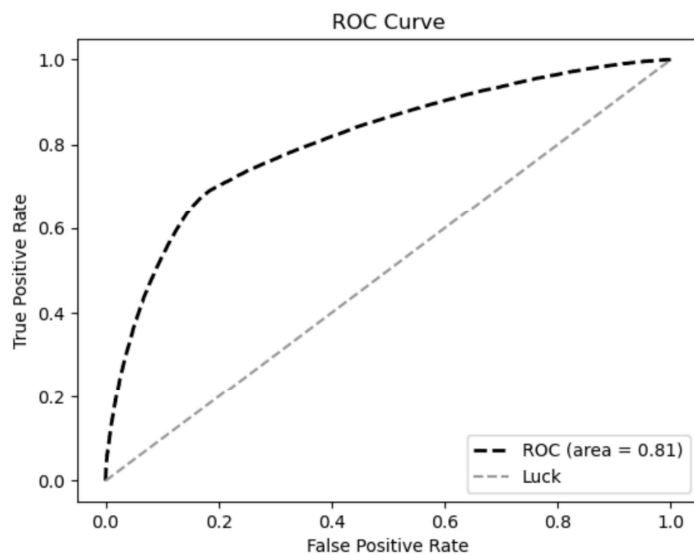
F1 score = 0.735582

测试集:

准确率: 0.752620

F1 score = 0.735660

AUC:0.807623



4个模型在各自验证集和测试集上的F1 score接近, 符合预期。

感知机的ROC曲线较平滑, SVM的ROC曲线出现了局部斜线。样本总体分布均匀。

AUC 均大于 0.8, 感知机的AUC略大于SVM。