# 机器学习

## 作业二

## 一. (30 points) 类别不平衡

信用卡欺诈检测数据集 (Credit Card Fraud Detection) 包含了 2013 年 9 月通过信用卡进行的欧洲持卡人的交易。这是一个非常典型的类别不平衡数据集，数据集中正常交易的标签远多于欺诈交易。请你根据附件中提供的该数据集完成以下问题:

1. 该数据集共有 284807 个样本，其中只有 492 个负样本。请按照训练集和测试集比例 4：1 的方式划分数据集（使用固定的随机种子)。在训练集上训练 SVM 模型，并计算该模型在测试集上的精度(如准确率、召回率、F1 分数，AUC 等)。请展示 SVM 模型训练过程的完整代码，并绘制 ROC 曲线（8 points)；

2. 请从上述训练集中的正样本中分别随机剔除 2000，20000，200000 个样本，剩余的正样本和训练集中原本的负样本共同组成新的训练集，测试集保持不变。请参照上一小问方式在这三个新的训练集上训练 svm 模型，并记录每个模型的精度。观察并比较这几组实验的结果，结合准确率与召回率的定义，请说明不平衡数据集对模型的影响（9 points)；

3. 除了上述第 2 问的随机欠采样的方式以外，对小类样本的“过采样”也是处理不平衡问题的基本策略。一种经典的方法为人工合成的过采样技术 (Synthetic Minority Over-sampling Technique, SMOTE), 其在合成样本时寻找小类中某一个样本的近邻, 并在该样本与近邻之间进行差值, 作为合成的新样本。请查阅相关资料，实现 SMOTE 算法中的 over sampling 函数, 以代码块的形式附于下方即可（8 points)；

```
"""
注意:
1. 这个框架提供了基本的结构，您需要完成所有标记为 'pass' 的函数。
2. 记得处理数值稳定性问题，例如在计算对数时避免除以零。
3. 在报告中详细讨论您的观察结果和任何有趣的发现。
"""
class SMOTE(object):
    def __init__(self , X, y, N, K, random_state =0):
        self.N = N # 每个小类样本合成样本个数
        self.K = K # 近邻个数
        self.label = y # 进行数据增强的类别
        self.sample = X
        self.n_sample , self.n = self.sample.shape # 获得样本个数，特征个数
```

```
14
15      def over_sampling(self):
16          pass
```

Listing 1: SMOTE 模型接口

4. 请说明 SMOTE 算法的缺点并讨论可能的改进方案（5 points）。

## 二. (20 points) 机器学习中的过拟合现象

本题以决策树和多层感知机为例, 探究机器学习中的过拟合现象。在教材 2.1 节中提到, 机器学习希望训练得到的模型在新样本上保持较好的泛化性能. 如果在训练集上将模型训练得“过好”, 捕获到了训练样本中对分类无关紧要的特性, 会导致模型难以泛化到未知样本上, 这种情况称为过拟合。

1. 请简要总结决策树和多层感知机的工作原理及其对应的缓解过拟合的手段（决策树相关原理可以参考教材第 4 章, 多层感知机可以参考教材第 5 章）(5 points)；

2. 请使用 scikit-learn 实现决策树模型, 并扰动决策树的最大深度 $max\_depth$, 一般来说, $max\_depth$ 的值越大, 决策树越复杂, 越容易过拟合, 实验并比较测试集精度, 讨论并分析观察到的过拟合现象等 (5 points)；

3. 对决策树算法的未剪枝、预剪枝和后剪枝进行实验比较, 并进行适当的统计显著性检验 (5 points)；

4. 请使用 PyTorch 或 scikit-learn 实现一个简单的多层感知机, 并通过层数、宽度或者优化轮数控制模型复杂度, 实验并比较测试集精度, 讨论并分析观察到的过拟合现象等（5 points)。

注: 请选择 1 至 3 个 UCI 机器学习库中的数据集进行实验。

# 三. (20 points) 激活函数比较

神经网络隐层使用的激活函数一般与输出层不同. 请画出以下几种常见的隐层激活函数的示意图并计算其导数（导数不存在时可指明）, 讨论其优劣（每小题 4 points）:

1. Sigmoid 函数, 定义如下

$$f(x) = \frac{1}{1 + e^{-x}}. \tag{1}$$

2. 双曲正切函数 (Hyperbolic Tangent Function, Tanh), 定义如下

$$f(x) = \tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}. \tag{2}$$

3. 修正线性函数 (Rectified Linear Unit, ReLU) 是近年来最为常用的隐层激活函数之一, 其定义如下

$$f(x) = \begin{cases} 0, & \text{if } x < 0; \\ x, & \text{otherwise.} \end{cases} \tag{3}$$

4. Softplus 函数, 定义如下

$$f(x) = \ln(1 + e^{x}). \tag{4}$$

5. Leaky Rectified Linear Unit (Leaky ReLU) 函数, 定义如下

$$f(x) = \begin{cases} \alpha x, & \text{if } x < 0; \\ x, & \text{otherwise.} \end{cases} \tag{5}$$

其中 $\alpha$ 为一个较小的正数, 比如 0.01.

# 四. (30 points) 神经网络实战

moons 是一个简单的二分类数据集，请实现一个简单的全连接神经网络，并参考教材图 5.8 实现反向传播算法训练该网络，用于解决二分类问题。

1. 使用 NumPy 手动实现神经网络和反向传播算法。(15 points)
2. 实现并比较不同的权重初始化方法。(5 points)
3. 在提供的 moons 数据集上训练网络，观察并分析收敛情况和训练过程。(10 points)

**提示:**

1. 神经网络实现：
   - 实现一个具有一个隐藏层的全连接神经网络。
   - 网络结构：输入层 (2 节点) → 隐藏层 (4 节点) → 输出层 (1 节点)
   - 隐藏层使用 ReLU 激活函数，输出层使用 Sigmoid 激活函数。
   - 使用交叉熵损失函数。
2. 权重初始化方法。实现以下三种初始化方法，并比较它们的性能：
   - 随机初始化：从均匀分布 $U(-0.5, 0.5)$ 中采样。
   - Xavier 初始化：根据前一层的节点数进行缩放。

     $$W_{ij} \sim U\left(-\frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}}, \frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}}\right)$$

     其中 $n_{in}$ 是前一层的节点数，$n_{out}$ 是当前层的节点数。
   - He 初始化：He 初始化假设每一层都是线性的，并且考虑了 ReLU 激活函数的特性。

     $$W_{ij} \sim N\left(0, \frac{2}{n_{in}}\right)$$

     其中 $n_{in}$ 是前一层的节点数。
3. 训练和分析：
   - 使用提供的 moons 数据集。
   - 实现小批量梯度下降算法。

- 记录并绘制训练过程中的损失值和准确率，训练结束后绘制决策边界。
- 比较不同初始化方法对训练过程和最终性能的影响并给出合理解释。
- 尝试不同的学习率，观察其对训练的影响并给出合理解释。

```
"""
注意：
1. 这个框架提供了基本的结构，您需要完成所有标记为 'pass' 的函数。
2. 确保正确实现前向传播、反向传播和梯度更新。
3. 在比较不同初始化方法时，保持其他超参数不变。
4. 记得处理数值稳定性问题，例如在计算对数时避免除以零。
5. 尝试使用不同的学习率（例如 0.01, 0.1, 1），并比较结果。
6. 在报告中详细讨论您的观察结果和任何有趣的发现。
"""
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split

# 生成数据集
X, y = make_moons(n_samples=1000, noise=0.1, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

class NeuralNetwork:
    def __init__(self, input_size, hidden_size, output_size, init_method='random'):
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size

        # 初始化权重和偏置
        if init_method == 'random':
            # 实现Random初始化
            pass
        elif init_method == 'xavier':
            # 实现Xavier初始化
            pass
        elif init_method == 'he':
            # 实现He初始化
            pass
        else:
            raise ValueError("Unsupported initialization method")

    def relu(self, x):
        return np.maximum(0, x)

    def sigmoid(self, x):
```

```
        return 1 / (1 + np.exp(-x))

    def forward(self, X):
        # 实现前向传播
        pass

    def backward(self, X, y, y_pred):
        # 实现反向传播
        pass

    def train(self, X, y, learning_rate, epochs, batch_size):
        # 实现小批量梯度下降训练
        pass

# 辅助函数
def plot_decision_boundary(model, X, y):
    # 绘制决策边界
    pass

def plot_training_process(losses, accuracies):
    # 绘制训练过程
    pass

# 主函数
def main():
    # 创建并训练模型
    # 绘制结果
    pass

if __name__ == "__main__":
    main()
```

Listing 2: 代码实现模板