



**《机器学习》**

**课程设计报告**

**报告题目：分类算法的对比学习研究**

**学院名称：计算机与人工智能学院**

**年 级： 2021级**

**授课教师：**  **詹大为**

**姓 名： 孙武周**

**学 号： 2021113501**

**二〇二三年六月**

**摘 要**：选用神经网络，逻辑回归，支持向量机三种方法进行分类算法的对比研究

**关键词**：OVR，NN，Logistic Regression ，SVM

1. **相关工作**

分类算法是机器学习中的一种常见类型，其主要任务是通过给定的特征数据，将输入数据划分为不同的类别。以下本文是三种分类算法的介绍：

1. 神经网络：神经网络是一种模仿人类大脑神经元网络的算法。它由神经元和它们之间的连接构成，可以进行监督学习或无监督学习。神经网络具有强大的泛化能力，可以学习和识别复杂模式，但需要较长的训练时间和复杂的架构设计。

2. 支持向量机：支持向量机（SVM）是一种非常常见的分类算法，它可以高效地处理高维数据，并且对数据的非线性转换有很强的适应性。SVM通过在特征空间中寻找支持向量来构建决策边界，从而实现分类任务。SVM的优点是泛化能力强，支持非线性分类，但需要选择合适的核函数。

1. 逻辑回归：逻辑回归是一种广泛应用于分类问题的线性模型。它通过拟合数据的概率分布，来预测某个类别的概率。逻辑回归非常简单，容易理解，训练速度快，并且可解释性高。其缺点是只能处理线性可分的数据，对异常值敏感。
   1. **基于神经网络（NN）的方法**

总体损失函数

神经网络的总损失等于所有单个数据点损失的平均。为了防止过拟合，通常还会在损失目标中增加对权重参数的正则化约束。因此，对于整个数据集的总损失函数为公式（1）

给定优化目标(W,b)后，梯度下降方法的每次迭代按如下公式（2）（3）分别更新参数w和b:

以下是两层神经网络基于均方损失函数的BP反向传播公式推导

公式（10）为均方损失函数

公式（10）为sigmoid激活函数,公式（11）为sigmoid函数的导函数

由公式（11）（14）（15）（16）更新w,b以及阈值

* 1. **基于逻辑回归（Logistic Regression）的方法**

极大似然函数推到得公式（18），变形得公式（19）

继续推导得

以下是本文所用梯度下降公式推导

**1.3基于线性支持向量机（LSVM）的方法**

公式（26）线性支持向量机，硬间隔

公式（27）软间隔支持向量机

其中c是一个超参数。c趋向于O的时候，允许无限大的误差，趋向于无穷大的时候，算法本质就是Hard，引入hinge损失后，优化目标Hard Margin SVM。

公式(28)是本文需用到的梯度下降

**2. 基础内容**

**2.1 OVR**

介绍该类方法如何实现手写体数字识别的，其优缺点是什么。请删除）

OVR指的是“One-vs-Rest”(一对多)，也称为“One-vs-All”。它是一种常用的多类别分类问题解决方法。在这种方法中，将n个类别的分类问题抽象为n个二分类问题，具体做法是对于每个类别，将该类别样本标记为正例，其余类别样本标记为负例，然后训练n个二分类器，每个分类器负责区分正例和本分类器对应的负例，以此实现多类别分类。

OVR是一种简单易懂的方法，适用于二分类算法，例如SVM、逻辑回归等。它的实现简单，只需要进行n次二分类即可。但在样本不平衡情况下，某些类别的样本数量较少会导致分类器的训练效果相对较差，此时可能需要使用其他的多类别分类方法，如OVO（One-vs-One）等。

**2.2 神经网络（NN）**

本文实现的神经网络可直接进行多分类，所以根据数据集的形状，定义input\_dim，hidden\_dim，output\_dim，其他超参数使用十折交叉验证法进行调参。本文所用其他超参数学习率lr=0.3,批大小batch\_size=4,迭代最大次数N=500.实现代码参考附录。

**2.3逻辑回归（Logistic Regression）**

逻辑回归是一种广泛应用于分类问题的线性模型。它通过拟合数据的概率分布，来预测某个类别的概率。单个逻辑回归分类器只能进行二分类，本文进行多分类采用OVR方法训练多个逻辑回归分类器。

**2.4 线性支持向量机（LSVM）**

线性支持向量机是寻找一个最优的超平面来将不同类别的样本分开，以达到分类的目的。单个的线性支持向量机分类器只能实现二分类，本文进行多分类采用OVR方法，训练多个线性支持向量机分类器。

**2.5 评估指标（准确率（accuracy））**

准确率（accuracy）是指分类器正确分类的样本数占总样本数的比例。通俗的解释，就是分类器正确分类的程度，它是一个分类器优劣性能的重要指标之一。

计算准确率的公式为：

准确率 = (TP + TN) / (TP + TN + FP + FN)

其中，TP表示真正例的数量，TN表示真反例的数量，FP表示假正例的数量，FN表示假反例的数量。

在二分类问题中，将样本分为正类和反类，那么准确率可以理解为整个分类器的预测正确率，即分类器预测为正类的样本中有多少是真正的正类（TP），分类器预测为反类的样本中有多少是真正的反类（TN）。

本文采用micro-accuracy来表示accuracy衡量各种分类算法的性能。micro-accuracy是一种常用的多类别分类性能评估指标，它是多类别分类的一些其他常用指标如准确率，召回率和F1值的一个变体。在多分类任务中，假设有K个类别，那么micro-accuracy可以定义为：将所有分类正确的样本数量（True Positives）累加起来，除以所有样本的数量，公式为：

micro-accuracy = (TP\_1 + TP\_2 + ... + TP\_K) / (N\_1 + N\_2 + ... + N\_K)

其中，TP\_k是第k个类别中分类正确的样本数量，N\_k是第k个类别中的所有样本数量。

micro-accuracy计算的是所有类别的预测正确样本总数占总样本数的比例，即对每个样本的分类结果都进行计算，每个样本的权重都是一样的。与传统的准确率不同，micro-accuracy更注重全局性能，即正确分类的样本数量在总样本数量中所占的比例。在多类别分类任务中，尤其是类别数量不平衡的情况下，micro-accuracy比起传统的准确率更能客观评价模型的性能。

**3. 运行步骤**

**3.1 数据清洗**

运行数据集目录下的 数据集转换和处理.py 程序即可将sklearn中的本文所用的数据集以及下载的data\_banknote\_authentication.txt 数据集转换成 csv格式并清洗掉数据集中损失数据和非法数据。运行 数据集划分.py 程序文件即可将数据集划分成训练集和测试集。

**3.2 参数调整**

调整k折交叉验证(调参)目录下的 k折交叉验证.py 程序里三种模型的超参数然后运行，不断调整运行直到找到三种模型合适的超参数。

**3.3 模型训练**

运行模型训练目录下的 模型训练.py 程序即可在六种数据集上训练三种模型，终端显示训练进度，训练结束后保存三种模型参数到模型参数目录下。

**3.4 模型测试**

运行模型测试目录下的 模型测试.py 程序即可得到三种模型在六种数据集上的测试结果，测试结果保存在测试结果目录下。

**4. 实验**

**4.1 数据集**

1. Iris数据集：鸢尾花数据集，包含150行，每行4个特征，分类为3类鸢尾花的品种之一。

2. Digits数据集：手写数字数据集，包含1797行，每行64个特征，分类为10个数字的之一。

3. Wine数据集：葡萄酒数据集，包含178行，每行13个特征，分类为3类葡萄酒中的一种。

4. Breast Cancer数据集：乳腺癌数据集，包含569行，每行30个特征，分类为良性或恶性。

5. Data Banknote Authentication数据集：货币鉴别数据集，包含1372行，每行4个特征，分类为真假货币之一。

6. Dermatology数据集：皮肤病数据集，包含358行，每行34个特征，分类为6类皮肤病之一。

**4.2 实验环境**

硬件环境

1. 计算机：LAPTOP-K5GRC2HU
2. CPU: 11th Gen Intel(R) Core(TM) i5-11400H @ 2.70GHz 2.69 GHz
3. RAM：16GB

软件环境

1. 操作系统：Windows10家庭中文版
2. 开发工具：Visual Studio Code

**4.3 实验结果**

表1 三种方法在六种数据集训练集上10折交叉验证平均准确率对比

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 数据集 | 神经网络 | OVR逻辑回归 | OVR支持向量机 |  |
| Iris | 0.93 | 0.69 | 0.62 |  |
| Digits | 0.96 | 0.96 | 0.88 |  |
| Wine | 0.97 | 0.92 | 0.97 |  |
| BC | 0.96 | 0.9 | 0.95 |  |
| DBA | 0.99 | 0.88 | 0.96 |  |
| Dermatology | 0.97 | 0.99 | 0.96 |  |

表2 三种方法在六种数据集测试集上准确率对比

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 优化器 | 神经网络 | OVR逻辑回归 | OVR支持向量机 |  |
| Iris | 0.93 | 0.73 | 0.71 |  |
| Digits | 0.96 | 0.93 | 0.83 |  |
| Wine | 1.0 | 0.98 | 0.98 |  |
| BC | 0.93 | 0.89 | 0.96 |  |
| DBA | 0.98 | 0.84 | 0.93 |  |
| Dermatology | 0.96 | 0.95 | 0.94 |  |

注：BC ，DBA分别为数据集Breast Cancer，Data Banknote Authentication。

**4.4 结果分析**

根据表格1 数据可得：

三种方法在六种数据集训练集上使用十折交叉验证方法除了Iris数据集得到较好的结果，这证明了在除Iris数据集上其他数据集超参数调整较好。对于Iris上的三种方法的超参数还需继续调整。

根据表格2数据可得：

1. 在Iris数据集中，神经网络方法的准确率与OVR逻辑回归相比仅稍高，而在Data Banknote Authentication数据集中表现最好。

2. 在Digits数据集中，神经网络方法表现最好，其准确率最高，OVR逻辑回归次之，OVR支持向量机表现最差。

3. 在Wine数据集中，三种方法表现都很好，但神经网络方法在该数据集上表现最佳，其准确率达到1.0。

4. 在BC数据集中，OVR支持向量机方法表现最好，神经网络方法和OVR逻辑回归方法均较差。

5. 在Data Banknote Authentication数据集中，神经网络表现最好，OVR逻辑回归最次。

6. 在Dermatology数据集中，三种方法的准确率相近，但神经网络和OVR逻辑回归略好于OVR支持向量机。

综上所述，神经网络方法在多数数据集上表现良好，具有较高的分类性能，但不是所有数据集都适用；OVR逻辑回归及OVR支持向量机方法在某些数据集上的表现也很好，因此在实践中需要根据实际情况选择合适的分类方法。

**5. 总结展望**

总结：本文采用三种分类方法进行对比学习研究，这三种方法分别是神经网络，逻辑回归，支持向量机，其中逻辑回归和支持向量机采用OVR方法进行多分类。三种方法都采用梯度下降算法进行优化，其中神经网络采用批量梯度下降，提高了训练的速度。最后三种方法在六种数据集上实验结果表现各异，其中神经网络在六种数据集测试效果均较好。

展望：本文不足之处分类算法的优化算法较为单一，其中支持向量机还可以采用SMO算法；使用的六种数据集多数数据集数据量较小；评价标准较为单一。后续将采用多样化的优化算法、数据量大的数据集和多样化的评价标准。

参考文献

1. Jabde, Meenal, et al. "Comparative Study of Machine Learning and Deep Learning Classifiers on Handwritten Numeral Recognition." International Symposium on Intelligent Informatics: Proceedings of ISI 2022. Singapore: Springer Nature Singapore, 2023.
2. Wei Wu, Xiaorong Gao and Shangkai Gao, "One-Versus-the-Rest(OVR) Algorithm: An Extension of Common Spatial Patterns(CSP) Algorithm to Multi-class Case," 2005 IEEE Engineering in Medicine and Biology 27th Annual Conference, Shanghai, China, 2005, pp. 2387-2390, doi: 10.1109/IEMBS.2005.1616947.

**附录**

**数据集处理相关代码：**

**数据集下载、清洗与格式转换：**

import pandas as pd

import sklearn.datasets as datasets

import numpy as np

# 加载数据集

iris = datasets.load\_iris()

digits = datasets.load\_digits()

wine = datasets.load\_wine()

breast\_cancer = datasets.load\_breast\_cancer()

# 读取csv格式数据

data\_banknote\_authentication\_df = pd.DataFrame(

    pd.read\_csv("./数据集//原始数据集/data\_banknote\_authentication.txt",

                sep=',',

                header=None), )

dermatology\_df = pd.DataFrame(

    pd.read\_csv("./数据集//原始数据集/archive/dermatology\_database\_1.csv"))

#数据清洗

dermatology\_df['age'] = dermatology\_df['age'].replace('?', np.nan)

dermatology\_df.dropna(subset=['age'], inplace=True)

dermatology\_df['class'] = dermatology\_df['class'].values.astype(float) - 1

# 将数据集转换为dataframe格式，同时保存标签

iris\_df = pd.DataFrame(data=iris.data, columns=iris.feature\_names)

iris\_df['target'] = iris.target

digits\_df = pd.DataFrame(data=digits.data)

digits\_df['target'] = digits.target

wine\_df = pd.DataFrame(data=wine.data, columns=wine.feature\_names)

wine\_df['target'] = wine.target

breast\_cancer\_df = pd.DataFrame(data=breast\_cancer.data,

                                columns=breast\_cancer.feature\_names)

breast\_cancer\_df['target'] = breast\_cancer.target

# 将dataframe保存为csv文件

iris\_df.to\_csv('./数据集//原始数据集/iris.csv', index=False)

digits\_df.to\_csv('./数据集//原始数据集/digits.csv', index=False)

wine\_df.to\_csv('./数据集//原始数据集/wine.csv', index=False)

breast\_cancer\_df.to\_csv('./数据集//原始数据集/breast\_cancer.csv', index=False)

data\_banknote\_authentication\_df.to\_csv(

    './数据集//原始数据集/data\_banknote\_authentication.csv', index=False)

dermatology\_df.to\_csv('./数据集//原始数据集/dermatology.csv', index=False)

**数据集划分：**

import pandas as pd

import random

import numpy as np

import os

def split\_dataset(dataset, ratio, random\_state=42):

    train\_size = int(len(dataset) \* ratio)

    train\_set = []

    test\_set = list(dataset)

    np.random.seed(random\_state)

    while len(train\_set) < train\_size:

        index = random.randrange(len(test\_set))

        train\_set.append(test\_set.pop(index))

    return pd.DataFrame(train\_set), pd.DataFrame(test\_set)

def split\_data(filename, dataset\_name, ratio=0.2):

    """

    读取CSV文件并将其分为训练集和测试集，然后将两个数据集保存为新的CSV文件

    :param filename: CSV文件名

    :param test\_size: 测试集所占比例，默认为0.2

    :return: 无返回值

    """

    # 读取CSV文件到pandas的DataFrame对象

    data\_df = pd.read\_csv(filename).values.astype(float)

    # 将数据集划分为训练集和测试集，test\_size表示测试集所占比例

    train\_df, test\_df = split\_dataset(data\_df, ratio=ratio)

    # 将训练集和测试集保存为CSV文件

    os.makedirs(f"数据集//训练集", exist\_ok=True)

    os.makedirs(f"数据集//测试集", exist\_ok=True)

    train\_df.to\_csv(f"数据集//训练集//{dataset\_name}.csv", index=False)

    test\_df.to\_csv(f"数据集//测试集//{dataset\_name}.csv", index=False)

if \_\_name\_\_ == '\_\_main\_\_':

    name\_list = [

        "breast\_cancer", "data\_banknote\_authentication", "dermatology",

        "digits", "iris", "wine"

    ]

    for dataset\_name in name\_list:

        dataset\_filename = f"数据集//原始数据集//{dataset\_name}.csv"

        #70%用作训练

        split\_data(dataset\_filename, dataset\_name, 0.7)

**模型代码：**

**NN：**

import numpy as np

import pandas as pd

import pickle

import os

#神经网络网络类

class Network:

    def \_\_init\_\_(self, input\_dim, hidden\_dim, output\_dim, lr) -> None:

        '''

        类初始化函数

        batch\_size  每次更新的数据量

        input\_dim   输入层大小

        hidden\_dim  隐藏层大小

        output\_dim  输出层大小

        activation  激活函数

        采用标准正态分布初始化模型参数

        '''

        self.input\_dim = input\_dim

        self.hidden\_dim = hidden\_dim

        self.output\_dim = output\_dim

        self.lr = lr

        self.weight1 = np.random.randn(input\_dim, hidden\_dim)

        self.bias1 = 0

        self.weight2 = np.random.randn(hidden\_dim, output\_dim)

        self.bias2 = 0

        self.output1 = 0

        self.output2 = 0

        #print(f"weight1shape={self.weight1.shape}")

    def reset(self):

        self.weight1 = np.random.randn(self.input\_dim, self.hidden\_dim)

        self.bias1 = 0

        self.weight2 = np.random.randn(self.hidden\_dim, self.output\_dim)

        self.bias2 = 0

        self.output1 = 0

        self.output2 = 0

    def classification(self, data):

        '''

        分类函数

        data 待分类数据

        返回该数据预测种类以及所有种类的分布

        '''

        self.output1 = self.activation(np.dot(data, self.weight1))

        self.output2 = self.activation(np.dot(self.output1, self.weight2))

        np.expand\_dims(np.argmax(self.output2, axis=1), axis=0)

        return np.expand\_dims(np.argmax(self.output2, axis=1),

                              axis=0).reshape(-1, 1), self.output2

        #权重更新函数

    def upadate(self, data, label):

        \_, \_ = self.classification(data)

        J, delta\_v, delta\_gamma, delta\_w, delta\_theta = self.loss\_function(

            data, label)

        self.weight1 = np.add(self.lr \* delta\_v, self.weight1)

        self.weight2 = np.add(self.lr \* delta\_w, self.weight2)

        return J

    #sigmoid激活函数

    def activation(self, x):

        return 1 / (np.exp(-x) + 1)

    def loss\_function(self, data, label):

        '''

        data   待分类数据

        label  数据标签，即该数据的种类

        返回损失函数以及各参数的梯度

        '''

        label\_array = np.zeros((data.shape[0], self.output\_dim))

        for (x, y) in [(i, int(label[i])) for i in range(data.shape[0])]:

            label\_array[x, y] = 1

        if label\_array.shape != self.output2.shape:

            print(

                f"label\_array.shape,self.output2.shape={label\_array.shape,self.output2.shape}"

            )

        g = self.output2 \* (1 - self.output2) \* (label\_array - self.output2)

        e = self.output1 \* (1 - self.output1) \* (np.dot(self.weight2, g.T)).T

        delta\_w = np.dot(self.output1.T, g)

        delta\_theta = -g  #隐藏层偏置值

        delta\_v = np.dot(np.array(data).T, e)

        delta\_gamma = -e  #输入层偏置值

        pred = self.output2

        J = 0.5 \* (label\_array - pred)\*\*2  #均方损失函数

        return J, delta\_v, delta\_gamma, delta\_w, delta\_theta

class Neural\_Network:

    '''

    input\_dim: 输入层大小

    hidden\_dim: 隐藏层大小

    output\_dim: 输出层大小

    lr: 学习率

    batch\_size: 批大小

    N: 训练轮数

    '''

    def \_\_init\_\_(self, input\_dim, hidden\_dim, output\_dim, lr, batch\_size,

                 N) -> None:

        self.model = Network(input\_dim, hidden\_dim, output\_dim, lr)

        self.batch\_size = batch\_size

        self.N = N

    #模型训练（测试）函数

    def train\_model(self, data, label):

        '''

        model    模型

        N        最大迭代次数

        lr       模型学习率

        data     训练数据

        label    训练数据对应的标签

        filename 保存（导入）模型参数的文件名

        train    True为训练模式，False为测试模式

        '''

        per\_loss = 0

        for single\_data, single\_label in zip(data, label):

            # print(label.shape)

            J = self.model.upadate(single\_data, single\_label)

            per\_loss += np.sum(J)

        return per\_loss

    def reset(self):

        self.model.reset()

    def fit(self, data, label):

        data = dataloader(data, self.batch\_size)

        label = dataloader(label, self.batch\_size)

        for \_ in range(self.N):

            self.train\_model(data, label)

    def predict(self, data):

        label\_pred, \_ = self.model.classification(data)

        return label\_pred

    def save\_params(self, filename):

        '''

        模型参数保存函数

        filename   模型参数保存路径

        '''

        path = os.makedirs(f"{filename}", exist\_ok=True)

        path = f"{filename}"

        with open(f"{path}\params.pkl", 'wb') as f:

            pickle.dump(self.model, f)

    def load\_params(self, filename):

        '''

        模型参数加载函数

        filename   模型参数加载路径

        '''

        path = os.makedirs(f"{filename}", exist\_ok=True)

        path = f"{filename}"

        with open(f"{path}\params.pkl", 'rb') as f:

            self.model = pickle.load(f)

#数据标准化函数

def data\_normalization(data):

    data\_copy = data.copy()

    m, n = data.shape

    for col in range(n):

        min\_value = np.min(data\_copy[:, col])

        max\_value = np.max(data\_copy[:, col])

        value = (max\_value - min\_value) if (max\_value - min\_value) != 0 else 1

        data\_copy[:, col] = (data\_copy[:, col] - min\_value) / value

    return data\_copy

#读取数据函数，返回属性和标签

def read\_data(filename):

    data = pd.read\_csv(filename).values.astype(float)

    return data[:, -1], data[:, :-1]

def dataloader(data, batch\_size):

    dataset = []

    for i in range(int(data.shape[0] / batch\_size) + 1):

        if (i + 1) \* batch\_size:

            dataset.append(np.array(data[i \* batch\_size:(i + 1) \* batch\_size]))

        else:

            dataset.append(np.array(data[i \* batch\_size:]))

    return dataset

if \_\_name\_\_ == '\_\_main\_\_':

    filename = "神经网络/dermatology"

    #训练

    loss = []

    train = True  #是否训练

    #读取数据以及数据标准化

    train\_filename = '数据集//数据集//训练集//dermatology.csv'

    test\_filename = '数据集//数据集//测试集//dermatology.csv'

    y\_train, x\_train = read\_data(train\_filename)

    y\_test, x\_test = read\_data(test\_filename)

    y\_train = y\_train.reshape((-1, 1))

    y\_test = y\_test.reshape((-1, 1))

    x\_train = data\_normalization(x\_train)

    x\_test = data\_normalization(x\_test)

    model = Neural\_Network(34, 22, 6, 0.3, 4, 500)

    model.fit(x\_train, y\_train)

    results = model.predict(x\_test)

    print(np.sum(results == y\_test) / y\_test.shape[0])

**Logistic Regression：**

代码1（保存为 逻辑回归.py）：

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

class Logistic:

    def \_\_init\_\_(self, feature\_size, alpha, N) -> None:

        #初始化系数矩阵,系数权重为1

        self.theta = np.ones((feature\_size, 1))

        #设置超参数

        self.alpha = alpha

        self.N = N

    #sigmoid函数

    def sigmoid(self, data):

        return (1 / (1 + np.exp(-data)))

    #模型预测函数

    def predict(self, data):

        #将大于0.5的变成1，小于0.5的变成0

        label\_pred = np.where(self.forward(data) > 0.5, 1, 0)

        return label\_pred

    #模型计算置信度

    def forward(self, data):

        label\_pos = self.sigmoid(np.dot(data, self.theta))

        return label\_pos

    #模型损失以及梯度计算函数

    def costFunction(self, data, label):

        m = len(label)

        #模型分类数据

        h = self.sigmoid(np.dot(data, self.theta))

        #将预测计算中不合格数据矫正

        one\_index, zero\_index = np.where(h >= 1), np.where(h <= 0)

        h[one\_index] = 1 - 1e-10

        h[zero\_index] = 1e-10

        #损失值

        loss = (-1 / m) \* np.sum(label \* np.log(h) +

                                 (1 - label) \* np.log(1 - h))

        #梯度

        grad = (1 / m) \* np.dot(data.T, (h - label))

        return loss, grad

    def fit(self, data, label):

        #开始使用梯度训练模型

        loss = []

        for i in range(self.N):

            per\_loss, grad = self.costFunction(data, label)

            self.theta = self.theta - self.alpha \* grad  #模型更新

            loss.append(per\_loss)

        return loss

#数据标准化函数

def data\_normlization(data):

    m, n = data.shape

    tempdata = data.copy()

    #对每一中属性的所有数据进行标准化

    for i in range(n):

        mu = np.mean(tempdata[:, i])

        sigma = np.std(tempdata[:, i])

        tempdata[:, i] = (tempdata[:, i] - mu) / sigma

    return tempdata

#读取数据函数，返回属性和标签

def read\_data(filename):

    data = pd.read\_csv(filename).values.astype(float)

    return data[:, -1], data[:, :-1]

if \_\_name\_\_ == '\_\_main\_\_':

    train\_filename = '数据集//训练集//breast\_cancer.csv'

    test\_filename = '数据集//测试集//breast\_cancer.csv'

    train\_label, train\_data = read\_data(train\_filename)

    test\_label, test\_data = read\_data(test\_filename)

    train\_label = train\_label.reshape((-1, 1))

    test\_label = test\_label.reshape((-1, 1))

    train\_data = data\_normlization(train\_data)

    test\_data = data\_normlization(test\_data)

    #改变训练集维度，为x加一维

    train\_data = np.concatenate((np.ones(

        (train\_data.shape[0], 1)), train\_data),

                                axis=1)

    test\_data = np.concatenate((np.ones((test\_data.shape[0], 1)), test\_data),

                               axis=1)

    data\_num = train\_data.shape[0]

    feature\_size = train\_data.shape[1]

    #设置超参数

    alpha = 0.1

    N = 500

    model = Logistic(feature\_size, alpha, N)

    loss = model.fit(train\_data, train\_label)

    results = model.predict(test\_data)

    print(np.sum(results == test\_label) / test\_label.shape[0])

    plt.plot(loss)

    plt.show()

代码2（保存为 OVR逻辑回归.py）：

from 逻辑回归 import Logistic

import numpy as np

import pandas as pd

import os

import pickle

#OVR多分类

class OVRLogistic:

    def \_\_init\_\_(self, input\_dim, output\_dim, alpha, N) -> None:

        self.output\_dim = output\_dim

        self.input\_dim = input\_dim

        self.alpha = alpha

        self.N = N

        self.models = [

            Logistic(input\_dim, alpha, N) for \_ in range(output\_dim)

        ]

    def reset(self):

        self.models = [

            Logistic(self.input\_dim, self.alpha, self.N)

            for \_ in range(self.output\_dim)

        ]

    def fit(self, data, label):

        for i, model in enumerate(self.models):

            label\_tran = label.copy()

            negative\_index = np.where(label == i)

            positive\_index = np.where(label != i)

            label\_tran[negative\_index] = 1

            label\_tran[positive\_index] = 0

            model.fit(data, label\_tran)

    def predict(self, data):

        label\_pred = []

        for single\_data in data:

            label = None

            label\_pos = 0

            for i, model in enumerate(self.models):

                # print(model.theta.shape)

                # print(data.shape)

                pos = model.forward(single\_data.reshape(1, -1)).item()

                if pos >= label\_pos:

                    label = i

                    label\_pos = pos

            label\_pred.append(label)

        return np.array(label\_pred).reshape(-1, 1)

    def save\_params(self, filename):

        '''

        模型参数保存函数

        filename   模型参数保存路径

        '''

        path = os.makedirs(f"{filename}", exist\_ok=True)

        path = f"{filename}"

        with open(f"{path}\params.pkl", 'wb') as f:

            pickle.dump(self.models, f)

    def load\_params(self, filename):

        '''

        模型参数加载函数

        filename   模型参数加载路径

        '''

        path = os.makedirs(f"{filename}", exist\_ok=True)

        path = f"{filename}"

        with open(f"{path}\params.pkl", 'rb') as f:

            self.models = pickle.load(f)

#数据标准化函数

def data\_normalization(data):

    data\_copy = data.copy()

    m, n = data.shape

    for col in range(n):

        min\_value = np.min(data\_copy[:, col])

        max\_value = np.max(data\_copy[:, col])

        value = (max\_value - min\_value) if (max\_value - min\_value) != 0 else 1

        data\_copy[:, col] = (data\_copy[:, col] - min\_value) / value

    return data\_copy

#读取数据函数，返回属性和标签

def read\_data(filename):

    data = pd.read\_csv(filename).values.astype(float)

    return data[:, -1], data[:, :-1]

if \_\_name\_\_ == '\_\_main\_\_':

    model = OVRLogistic(34, 6, 1, 500)

    train\_filename = '数据集//训练集//dermatology.csv'

    test\_filename = '数据集//测试集//dermatology.csv'

    train\_label, train\_data = read\_data(train\_filename)

    test\_label, test\_data = read\_data(test\_filename)

    train\_label = train\_label.reshape((-1, 1))

    test\_label = test\_label.reshape((-1, 1))

    train\_data = data\_normalization(train\_data)

    test\_data = data\_normalization(test\_data)

    model.load\_params('模型参数//OVR逻辑回归//dermatology')

    model.fit(train\_data, train\_label)

    # model.save\_params('模型参数//OVR逻辑回归//dermatology')

    results = model.predict(test\_data)

    print(np.sum(results == test\_label) / test\_label.shape[0])

**SVM：**

代码1（保存为 支持向量机.py）：

import random

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

from matplotlib.font\_manager import FontProperties

# 设置 Matplotlib 的默认字体

plt.rcParams['font.family'] = 'sans-serif'

plt.rcParams['font.sans-serif'] = ['SimHei']

class SVM:

    def \_\_init\_\_(self, feature\_size, C, alpha, N) -> None:

        '''

        w 超平面权重

        feature\_size 特征大小, 即属性的个数

        b 偏置值

        alpha 学习率

        C 惩罚系数

        N 最大迭代次数

        '''

        self.feature\_size = feature\_size

        self.C = C

        self.alpha = alpha

        self.w = np.zeros(feature\_size)

        self.b = 0

        self.N = N

    def forward(self, x):

        return np.dot(x, self.w) + self.b

    def predict(self, x):

        y = self.forward(x)

        y\_pred = y.copy()

        negative\_index = np.where(y > 0)

        positive\_index = np.where(y < 0)

        y\_pred[negative\_index] = 1

        y\_pred[positive\_index] = -1

        return np.array([y\_pred]).reshape(-1, 1)

    #更新参数函数

    def train(self, data, label):

        # 随机梯度下降

        # r = random.randint(0, self.feature\_size - 1)

        # predict = self.forward(data[r])

        # if label[r] \* predict < 1:

        #     self.w = self.w - self.alpha \* (self.w -

        #                                     self.C \* label[r] \* data[r])

        #     self.b = self.b - self.alpha \* (-label[r] \* self.C)

        # else:

        #     self.w = self.w - self.w \* self.alpha

        #梯度下降

        for r in range(data.shape[0]):

            predict = self.forward(data[r])

            if label[r] \* predict < 1:

                self.w = self.w - self.alpha \* (self.w -

                                                self.C \* label[r] \* data[r])

                self.b = self.b - self.alpha \* (-label[r] \* self.C)

            else:

                self.w = self.w - self.w \* self.alpha

    #训练函数，返回权重

    def fit(self, data, label):

        for \_ in range(self.N):

            self.train(data, label)

#读取数据函数，返回属性和标签

def read\_data(filename):

    data = pd.read\_csv(filename).values.astype(float)

    return data[:, -1], data[:, :-1]

if \_\_name\_\_ == '\_\_main\_\_':

    #读取数据

    train\_filename = '数据集//训练集//breast\_cancer.csv'

    test\_filename = '数据集//测试集//breast\_cancer.csv'

    train\_label, train\_data = read\_data(train\_filename)

    test\_label, test\_data = read\_data(test\_filename)

    train\_label = train\_label.reshape((-1, 1))

    test\_label = test\_label.reshape((-1, 1))

    model = SVM(30, 100, 0.0001, 2000)

    model.fit(train\_data, train\_label)

    results = model.predict(test\_data)

    print(np.sum(results == test\_label) / test\_label.shape[0])

代码2（保存为 OVR支持向量机.py）：

from 支持向量机 import SVM

import pandas as pd

import numpy as np

import os

import pickle

#OVR多分类

class OVRSVM:

    def \_\_init\_\_(self, input\_dim, output\_dim, C, alpha, N) -> None:

        self.input\_dim = input\_dim

        self.output\_dim = output\_dim

        self.alpha = alpha

        self.C = C

        self.N = N

        self.models = [SVM(input\_dim, C, alpha, N) for \_ in range(output\_dim)]

    def reset(self):

        self.models = [

            SVM(self.input\_dim, self.C, self.alpha, self.N)

            for \_ in range(self.output\_dim)

        ]

    def fit(self, data, label):

        for i, model in enumerate(self.models):

            label\_tran = label.copy()

            #print(label\_tran)

            negative\_index = np.where(label == i)

            positive\_index = np.where(label != i)

            label\_tran[negative\_index] = 1

            label\_tran[positive\_index] = -1

            model.fit(data, label\_tran)

    def predict(self, data):

        label\_pred = []

        for single\_data in data:

            label = None

            label\_pos = 0

            for i, model in enumerate(self.models):

                pos = model.forward(single\_data.reshape(1, -1)).item()

                if pos >= label\_pos:

                    label = i

                    label\_pos = pos

            label\_pred.append(label)

        return np.array(label\_pred).reshape(-1, 1)

    def save\_params(self, filename):

        '''

        模型参数保存函数

        filename   模型参数保存路径

        '''

        path = os.makedirs(f"{filename}", exist\_ok=True)

        path = f"{filename}"

        with open(f"{path}\params.pkl", 'wb') as f:

            pickle.dump(self.models, f)

    def load\_params(self, filename):

        '''

        模型参数加载函数

        filename   模型参数加载路径

        '''

        path = os.makedirs(f"{filename}", exist\_ok=True)

        path = f"{filename}"

        with open(f"{path}\params.pkl", 'rb') as f:

            self.models = pickle.load(f)

#数据标准化函数

def data\_normalization(data):

    data\_copy = data.copy()

    m, n = data.shape

    for col in range(n):

        min\_value = np.min(data\_copy[:, col])

        max\_value = np.max(data\_copy[:, col])

        value = (max\_value - min\_value) if (max\_value - min\_value) != 0 else 1

        data\_copy[:, col] = (data\_copy[:, col] - min\_value) / value

    return data\_copy

#读取数据函数，返回属性和标签

def read\_data(filename):

    data = pd.read\_csv(filename).values.astype(float)

    return data[:, -1], data[:, :-1]

if \_\_name\_\_ == '\_\_main\_\_':

    # train\_filename = '数据集\data\_banknote\_authentication.csv'

    # test\_filename = '数据集\data\_banknote\_authentication.csv'

    train\_filename = '数据集//训练集//dermatology.csv'

    test\_filename = '数据集//测试集//dermatology.csv'

    model = OVRSVM(34, 6, 100, 0.0001, 3000)

    # train\_filename = 'experiment\_05\_training\_set.csv'

    # test\_filename = 'experiment\_05\_testing\_set.csv'

    train\_label, train\_data = read\_data(train\_filename)

    test\_label, test\_data = read\_data(test\_filename)

    train\_label = train\_label.reshape((-1, 1))

    test\_label = test\_label.reshape((-1, 1))

    train\_data = data\_normalization(train\_data)

    test\_data = data\_normalization(test\_data)

    model.fit(train\_data, train\_label)

    model.save\_params('OVR支持向量机//dermatology')

    model.load\_params('OVR支持向量机//dermatology')

    results = model.predict(test\_data)

    print(np.sum(results == test\_label) / test\_label.shape[0])

**10折交叉验证（调参）代码：**

import numpy as np

import pandas as pd

import os

import sys

sys.path.append('模型文件')

from OVR逻辑回归 import read\_data, data\_normalization, OVRLogistic

from OVR支持向量机 import OVRSVM

from 神经网络 import Neural\_Network

#返回accuracy最高的模型

def k\_fold\_cross\_validation(

    model,

    X,

    y,

    result\_filename,

    k=5,

    random\_state=42,

):

    # 对索引数组进行随机洗牌

    np.random.seed(random\_state)

    indices = np.arange(len(X))

    np.random.shuffle(indices)

    #数据集随机化

    X = X[indices]

    y = y[indices]

    # 将数据均分为k个段

    subset\_size = len(X) // k

    subsets\_X = [X[i:i + subset\_size] for i in range(0, len(X), subset\_size)]

    subsets\_y = [y[i:i + subset\_size] for i in range(0, len(y), subset\_size)]

    #寻找最优准确率和最优模型

    best\_accuracy = 0

    worst\_accuracy = 1

    average\_accuracy = 0

    best\_model = None

    # 进行k次循环，每一次取出第k份为验证集，其余为训练集

    for i in range(k):

        X\_train, y\_train = [], []

        for j in range(k):

            if j != i:

                X\_train.append(subsets\_X[j])

                y\_train.append(subsets\_y[j])

        X\_train = np.array(np.concatenate(X\_train))

        X\_test, y\_test = subsets\_X[i], subsets\_y[i]

        y\_test = y\_test.reshape(-1, 1)

        y\_train = np.concatenate(y\_train).reshape(-1, 1)

        model.fit(X\_train, y\_train)

        results = model.predict(X\_test)

        accuracy = (np.sum(results == y\_test) / y\_test.shape[0])

        steps = 10

        #寻找最优模型

        worst\_accuracy = min(accuracy, worst\_accuracy)

        average\_accuracy += (accuracy - average\_accuracy) / (i + 1)

        if accuracy > best\_accuracy:

            best\_model = model

            best\_accuracy = accuracy

        print(f"step={i},accuracy={round(accuracy,2)}")

        #运行结果写入文件

        with open(result\_filename, "a") as f:

            f.write(f"step={i},accuracy={round(accuracy,2)}\n")

        #模型重置

        model.reset()

    #返回最优模型

    with open(result\_filename, "a") as f:

        f.write(

            f"best\_accuracy={round(best\_accuracy,2)}\nworst\_accuracy={round(worst\_accuracy,2)}\naverage\_accuracy={round(average\_accuracy,2)}\n\n"

        )

    print()

if \_\_name\_\_ == '\_\_main\_\_':

    name\_list = [

        "iris", "digits", "wine", "breast\_cancer",

        "data\_banknote\_authentication", "dermatology"

    ]

    for dataset\_name in name\_list:

        print(f"基于数据集{dataset\_name}的验证")

        dataset\_filename = f"数据集//训练集//{dataset\_name}.csv"

        result\_filename = f"k折交叉验证(调参)\\交叉验证结果"

        label, data = read\_data(dataset\_filename)

        data = data\_normalization(data)

        label\_df = pd.DataFrame(label.copy())

        label = label.reshape(-1, 1)

        result\_path = os.makedirs(f"{result\_filename}", exist\_ok=True)

        result\_path = result\_filename

        result\_filename = f"{result\_filename}\{dataset\_name}.txt"

        with open(result\_filename, "w") as f:

            f.write(f"神经网络:\n")

        print("神经网络:")

        param\_filename = f"模型参数//神经网络//{dataset\_name}"

        input\_dim = data.shape[1]

        hidden\_dim = data.shape[1]

        output\_dim = len(label\_df.iloc[:, -1].unique())

        lr = 0.3

        batch\_size = 4

        N = 500

        model = Neural\_Network(input\_dim, hidden\_dim, output\_dim, lr,

                               batch\_size, N)

        k\_fold\_cross\_validation(

            model,

            data,

            label,

            result\_filename,

            10,

        )

        model.fit(data, label)

        model.save\_params(param\_filename)

        with open(result\_filename, "a") as f:

            f.write(f"OVR逻辑回归:\n")

        print("OVR逻辑回归:")

        param\_filename = f"模型参数//OVR逻辑回归//{dataset\_name}"

        alpha = 1

        N = 500

        model = OVRLogistic(input\_dim, output\_dim, alpha, N)

        k\_fold\_cross\_validation(model, data, label, result\_filename, 10)

        model.fit(data, label)

        model.save\_params(param\_filename)

        with open(result\_filename, "a") as f:

            f.write(f"OVR支持向量机:\n")

        print("OVR支持向量机:")

        param\_filename = f"模型参数//OVR支持向量机//{dataset\_name}"

        C = 100

        alpha = 0.0001

        N = 500

        model = OVRSVM(input\_dim, output\_dim, C, alpha, N)

        k\_fold\_cross\_validation(model, data, label, result\_filename, 10)

        model.fit(data, label)

        model.save\_params(param\_filename)

**模型训练代码：**

import numpy as np

import pandas as pd

import os

import sys

sys.path.append('模型文件')

from OVR逻辑回归 import read\_data, data\_normalization, OVRLogistic

from OVR支持向量机 import OVRSVM

from 神经网络 import Neural\_Network

if \_\_name\_\_ == '\_\_main\_\_':

    name\_list = [

        "iris", "digits", "wine", "breast\_cancer",

        "data\_banknote\_authentication", "dermatology"

    ]

    for dataset\_name in name\_list:

        dataset\_filename = f"数据集//训练集//{dataset\_name}.csv"

        #读取数据并调整数据形状

        label, data = read\_data(dataset\_filename)

        data = data\_normalization(data)

        label\_df = pd.DataFrame(label.copy())

        label = label.reshape(-1, 1)

        print(f"基于完整训练集{dataset\_name}开始训练模型..")

        print("神经网络...")

        param\_filename = f"模型参数//神经网络//{dataset\_name}"

        input\_dim = data.shape[1]

        hidden\_dim = data.shape[1]

        output\_dim = len(label\_df.iloc[:, -1].unique())

        lr = 0.3

        batch\_size = 4

        N = 500

        model = Neural\_Network(input\_dim, hidden\_dim, output\_dim, lr,

                               batch\_size, N)

        model.fit(data, label)

        model.save\_params(param\_filename)

        print("OVR逻辑回归..")

        param\_filename = f"模型参数//OVR逻辑回归//{dataset\_name}"

        alpha = 1

        N = 500

        model = OVRLogistic(input\_dim, output\_dim, alpha, N)

        model.fit(data, label)

        model.save\_params(param\_filename)

        print("OVR支持向量机..")

        param\_filename = f"模型参数//OVR支持向量机//{dataset\_name}"

        C = 100

        alpha = 0.0001

        N = 500

        model = OVRSVM(input\_dim, output\_dim, C, alpha, N)

        model.fit(data, label)

        model.save\_params(param\_filename)

        print("\n")

**模型测试代码：**

import numpy as np

import pandas as pd

import sys

sys.path.append('模型文件')

from OVR逻辑回归 import read\_data, data\_normalization, OVRLogistic

from OVR支持向量机 import OVRSVM

from 神经网络 import Neural\_Network

if \_\_name\_\_ == '\_\_main\_\_':

    name\_list = [

        "iris", "digits", "wine", "breast\_cancer",

        "data\_banknote\_authentication", "dermatology"

    ]

    result\_filename = "模型测试"

    result\_filename = f"{result\_filename}//测试结果//result.txt"

    with open(result\_filename, "w") as f:

        f.write(f"测试结果:\n")

    for dataset\_name in name\_list:

        dataset\_filename = f"数据集//测试集//{dataset\_name}.csv"

        params\_filename = f"模型参数"

        label, data = read\_data(dataset\_filename)

        data = data\_normalization(data)

        label\_df = pd.DataFrame(label.copy())

        label = label.reshape(-1, 1)

        print(f"基于测试集{dataset\_name}开始测试模型..")

        with open(result\_filename, "a") as f:

            f.write(f"基于测试集{dataset\_name}开始测试模型..\n")

        print("神经网络:", end=' ')

        input\_dim = data.shape[1]

        hidden\_dim = data.shape[1]

        output\_dim = len(label\_df.iloc[:, -1].unique())

        lr = 0.3

        batch\_size = 4

        N = 500

        model = Neural\_Network(input\_dim, hidden\_dim, output\_dim, lr,

                               batch\_size, N)

        model.load\_params(f"{params\_filename}//神经网络//{dataset\_name}")

        results = model.predict(data)

        accuracy = (np.sum(results == label) / label.shape[0])

        print(round(accuracy, 2))

        with open(result\_filename, "a") as f:

            f.write(f"神经网络: {round(accuracy,2)}\n")

        print("OVR逻辑回归:", end=' ')

        alpha = 1

        N = 500

        model = OVRLogistic(input\_dim, output\_dim, alpha, N)

        model.load\_params(f"{params\_filename}//OVR逻辑回归//{dataset\_name}")

        results = model.predict(data)

        accuracy = (np.sum(results == label) / label.shape[0])

        print(round(accuracy, 2))

        with open(result\_filename, "a") as f:

            f.write(f"OVR逻辑回归: {round(accuracy,2)}\n")

        print("OVR支持向量机:", end=' ')

        C = 100

        alpha = 0.0001

        N = 500

        model = OVRSVM(input\_dim, output\_dim, C, alpha, N)

        model.load\_params(f"{params\_filename}//OVR支持向量机//{dataset\_name}")

        results = model.predict(data)

        accuracy = (np.sum(results == label) / label.shape[0])

        print(round(accuracy, 2))

        with open(result\_filename, "a") as f:

            f.write(f"OVR支持向量机: {round(accuracy,2)}\n\n")

        print("\n")