实验 2　使用 python 实现线性 SVM 算法

姓名：李坤璘

班级学号：20 智能 03 2019202216

一、 实验目的：

1. 掌握线性 SVM 算法的 python 实现；

二、 实验条件：

1. PC 微机一台和 Python 环境。

三、 实验原理：

1、线性可分SVM

•输入：线性可分训练数据集$T=\{(x_1,y_1),(x_2,y_2),...,(x_N,y_N)\}$

$$x_i\in\mathcal{X}=R^n, y_i\in\mathcal{Y}=\{+1,-1\}, i=1,2,...,N$$

•输出：最大间隔分离超平面和分类决策函数

•1、构造并求解约束最优化问题

$$\min_{\alpha}\ \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j(x_i\cdot x_j)-\sum_{i=1}^{N}\alpha_i$$

$$s.t.\ \sum_{i=1}^{N}\alpha_i y_i=0$$

$$\alpha_i\geq 0,\ i=1,2,...,N$$

• 求得最优解：

$$\alpha^*=(\alpha_1^*,\alpha_2^*,...,\alpha_l^*)^T$$

2、计算$w^*=\sum_{i=1}^{N}\alpha_i^* y_i x_i$

并选择α*的一个正分量 $\alpha_j^*>0$，计算

$$b^*=y_j-\sum_{i=1}^{N}\alpha_i^* y_i(x_i\cdot x_j)$$

3、求得分离超平面 $w^*\cdot x+b^*=0$

分类决策函数 $f(x)=sign(w^*\cdot x+b^*)$

2、线性不可分线性支持向量机学习算法

•输入：线性不可分训练数据集 $T=\{(x_1,y_1),(x_2,y_2),...,(x_N,y_N)\}$

$$x_i \in \mathcal{X}=R^n, y_i \in \mathcal{Y}=\{-1,+1\}, i=1,2,...,N$$

•输出：分离超平面和分类决策函数

1、构造并求解约束最优化问题

$$\min_{\alpha} \ \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j (x_i \cdot x_j) - \sum_{i=1}^{N}\alpha_i$$

$$s.t. \quad \sum_{i=1}^{N}\alpha_i y_i = 0$$

$$0 \le \alpha_i \le C, \ i=1,2,...,N$$

求得最优解：

$$\alpha^* = (\alpha_1^*, \alpha_2^*, ..., \alpha_l^*)^T$$

2、计算 $w^* = \sum_{i=1}^{N}\alpha_i^* y_i x_i$

并选择α*，适合条件 $0 < \alpha_j^* < C$ ，计算

$$b^* = y_j - \sum_{i=1}^{N}\alpha_i^* y_i (x_i \cdot x_j)$$

3、求得分离超平面 $w^* \cdot x + b^* = 0$

分类决策函数 $f(x) = sign(w^* \cdot x + b^*)$

四、 实验内容：

1、随机产生两类样本数据：

X=$\begin{bmatrix} x1 \\ x2 \end{bmatrix}$，x1 和 x2 从多元正态分布中随机生成，y={-1,1}

画出样本图。

2、梯度下降法更新优化变量 w 和 b

3、得到分类决策函数 f(x）=sign(wx+b)

五、实验代码及结果

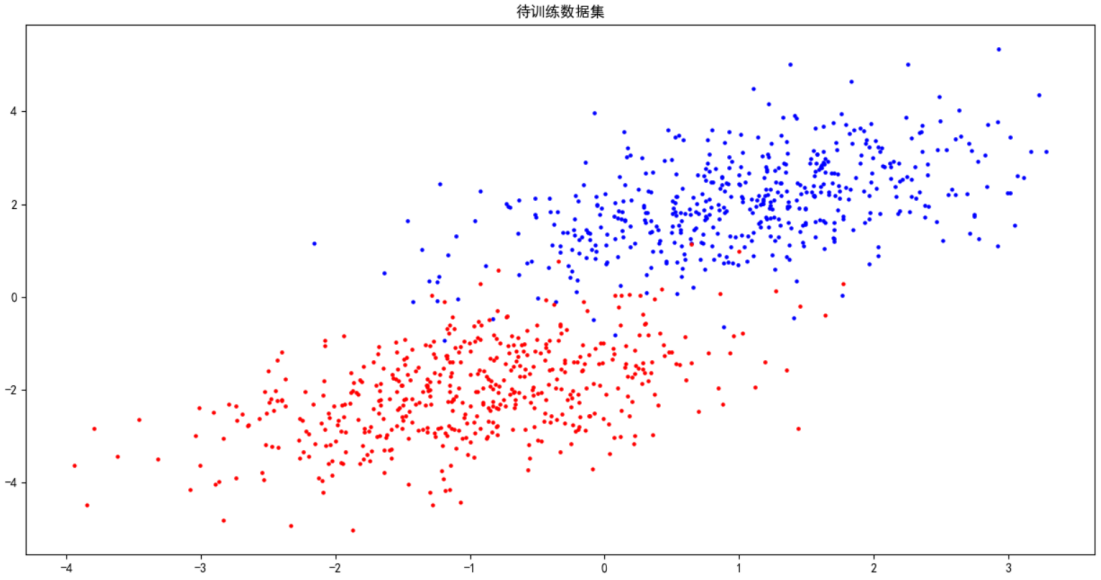
```
import numpy as np
from matplotlib import pyplot as plt

plt.rcParams['font.sans-serif'] = ['SimHei']  # 用来正常显示中文标签
plt.rcParams['axes.unicode_minus'] = False  # 用来正常显示负号

'''1.随机生成两类样本数据'''
mean1 = np.array([1, 2])  # 样本数据的平均值
cov1 = np.array([[1, 0.5], [0.5, 1]])  # 协方差矩阵
mean2 = np.array([-1, -2])
cov2 = np.array([[1, 0.5], [0.5, 1]])

Sum = 500
x1 = np.random.multivariate_normal(mean=mean1, cov=cov1,
size=Sum)  # 正态分布生成 500 个随机样本数据
x2 = np.random.multivariate_normal(mean=mean2, cov=cov2,
size=Sum)

'''2.为两类样本标记分类'''
y1 = np.ones((Sum,))  # 第一类样本的类别标签 y 都为 1
y2 = -np.ones((Sum,))  # 第二类样本的类别标签 y 都为-1
x = np.concatenate((x1, x2), axis=0)  # 将两类样本数据并在一起
y = np.concatenate((y1, y2), axis=0)  # 将两类样本数据的类别标签并在一
起

'''3.可视化样本数据'''
plt.figure(1)  # 创建一个新的图形窗口
plt.scatter(x1[:, 0], x1[:, 1], c='b', s=5)  # 绘制第一类样本
plt.scatter(x2[:, 0], x2[:, 1], c='r', s=5)  # 绘制第二类样本
plt.title('待训练数据集')

'''4.定义线性 SVM'''
# 初始化参数
w = np.zeros(x.shape[1])  # 参数 w 初始化为全零向量，长度为样本特征的维数
b = 0  # 参数 b 初始化为 0
num_iteration = 10000  # 迭代次数
learning_rate = 0.01  # 学习率

# 定义线性 SVM 模型
def svm(x, w, b):
    return np.dot(x, w) + b
```

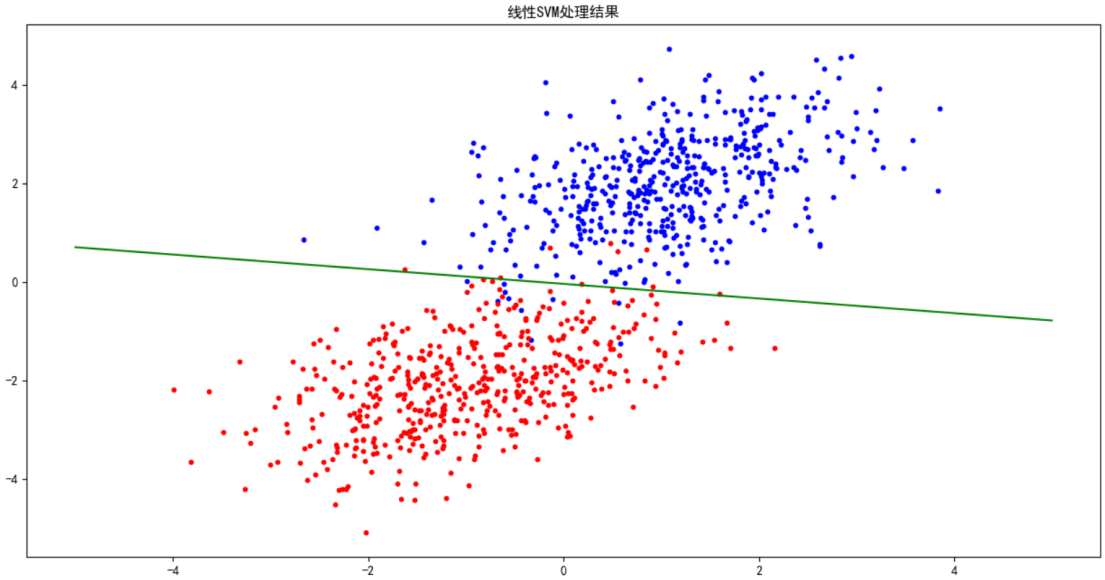
```
# 定义损失函数
def loss(x, y, w, b):
    N = x.shape[0]  # 样本数量
    margin = y * svm(x, w, b)  # 计算每个样本到分离超平面的距离，并与样本类别标签相乘
    loss = 1 - margin  # 计算损失
    loss[loss < 0] = 0  # 如果距离大于等于1，则损失为0；否则损失为1-距离
    return np.sum(loss) / N  # 计算平均损失

# 计算梯度
def grad(x, y, w, b):
    N = x.shape[0]  # 样本数量
    margin = y * svm(x, w, b)  # 计算每个样本到分离超平面的距离，并与样本类别标签相乘
    grad_w = np.zeros(x.shape[1])  # 初始化w的梯度为全零向量，长度等于样本特征的维数
    grad_b = 0  # 初始化b的梯度为0
    for i in range(N):
        if margin[i] < 1:  # 如果距离小于1，即样本被分类错误
            grad_w += -y[i] * x[i]  # 更新w的梯度
            grad_b += -y[i]  # 更新b的梯度
    grad_w /= N  # 计算w的平均梯度
    grad_b /= N  # 计算b的平均梯度
    return grad_w, grad_b  # 返回梯度

'''5.迭代训练'''
for i in range(num_iteration):
    # 计算梯度
    grad_w, grad_b = grad(x, y, w, b)
    # 更新参数
    w -= learning_rate * grad_w
    b -= learning_rate * grad_b
    # 计算损失
    train_loss = loss(x, y, w, b)
    if i % 500 == 0:
        print('迭代次数 {}/{}: 目前损失率 = {}'.format(i,
num_iteration, train_loss))

'''6.绘制超平面'''
plt.figure(2)  # 创建一个新的图形窗口
plt.scatter(x1[:, 0], x1[:, 1], c='b', s=10)  # 绘制第一类样本
plt.scatter(x2[:, 0], x2[:, 1], c='r', s=10)  # 绘制第二类样本
x_axis = np.linspace(-5, 5, 100)  # 在x轴上生成100个点
y_axis = -(w[0] * x_axis + b) / w[1]  # 计算超平面在x轴上的对应点
```

```
plt.plot(x_axis, y_axis, c='g')  # 绘制超平面
plt.title('线性 SVM 处理结果')  # 设置图像标题

'''7.计算准确率'''
y_pred = np.sign(svm(x, w, b))  # 预测样本类别
accuracy = np.mean(np.equal(y_pred, y))  # 计算准确率
print('最终训练准确率 = {}'.format(accuracy))  # 打印准确率
plt.show()  # 显示图像
```

```
F:\anaconda_env\Tensorflow26\python.exe "F:/Code/Machine Learning&Deep Learning/2-SVM/Linear.py"
迭代次数 0/10000: 目前损失率 = 0.9480574043657922
迭代次数 500/10000: 目前损失率 = 0.06985312809171317
迭代次数 1000/10000: 目前损失率 = 0.06413889651047837
迭代次数 1500/10000: 目前损失率 = 0.06082590040324995
迭代次数 2000/10000: 目前损失率 = 0.059263592454057344
迭代次数 2500/10000: 目前损失率 = 0.05834132016715169
迭代次数 3000/10000: 目前损失率 = 0.057725417414158285
迭代次数 3500/10000: 目前损失率 = 0.05722545975235629
迭代次数 4000/10000: 目前损失率 = 0.056790757265929775
迭代次数 4500/10000: 目前损失率 = 0.05646153806077743
迭代次数 5000/10000: 目前损失率 = 0.05622345681586361
迭代次数 5500/10000: 目前损失率 = 0.05601145169261112
迭代次数 6000/10000: 目前损失率 = 0.05585714060988394
迭代次数 6500/10000: 目前损失率 = 0.05574418213309295
迭代次数 7000/10000: 目前损失率 = 0.055650026941568974
迭代次数 7500/10000: 目前损失率 = 0.05555993103579314
迭代次数 8000/10000: 目前损失率 = 0.055471430665335705
迭代次数 8500/10000: 目前损失率 = 0.055395301804179835
迭代次数 9000/10000: 目前损失率 = 0.05535878467545989
迭代次数 9500/10000: 目前损失率 = 0.05532226754673994
最终训练准确率 = 0.976
```

如图所示，在训练样本集的时候损失率是不断变化的，且随着迭代次数的增加，损失率即错误率总体呈现下降趋势，说明随着梯度下降其 w 和 b 在不断向最准确的结果收敛。最终训练准确率也比较可观，因为是线性 SVM 且生成的样本较为随机，所以不能保证全部分类正确，不过这样也避免了过拟合的现象。
样本集和训练出的超平面如下图所示：





六、实验总结

这个实验演示了如何通过 Python 使用线性 SVM 算法对二元分类问题进行建模和求解。首先，我们使用多元正态分布函数，生成两类随机的样本数据，并将它们可视化，可以使用 numpy 和 matplotlib.pyplot 库完成这项任务。

接着我们定义一个支持向量机（SVM）函数框架，并使用梯度下降法更新优化变量 W 和 b。其中学习率需要选取合适的值，以达到最优的分类效果，并在 loss 函数中迭代训练数据多次，根据每个数据点的情况来更新 W 和 b。

在训练完成后，我们使用定义好的决策函数，对数据进行分类决策。其中，决策函数根据 W 和 b 得出样本在超平面的投影，再根据投影的结果返回预测的类别。

最后，为了评估分类器的效果，我们计算分类器的准确率验证分类器的泛化能力。

总的来说，实现 SVM 分类器需要掌握多元正态分布函数、梯度下降法、支持向量机的决策函数等知识点。在实际应用中，我们还需要调参、处理数据预处理等操作，以至于该 demo 模型能够应用在不同的领域，达到最佳的分类效果。