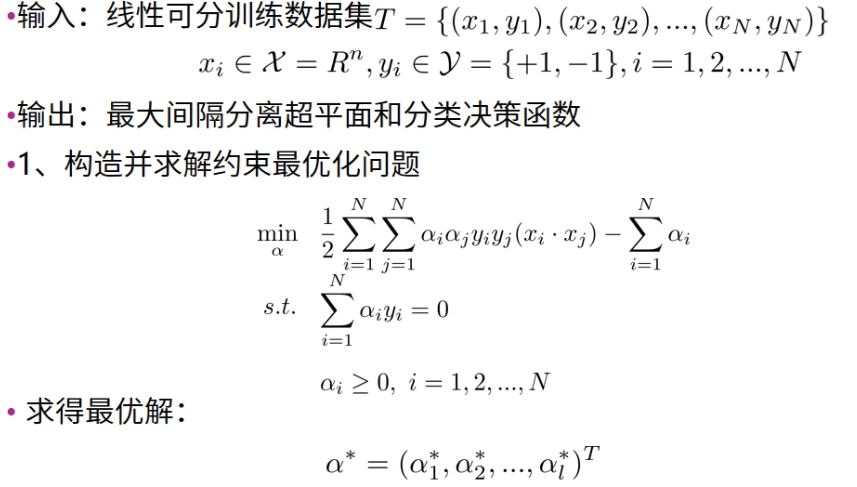
**实验2 使用python实现线性SVM算法**

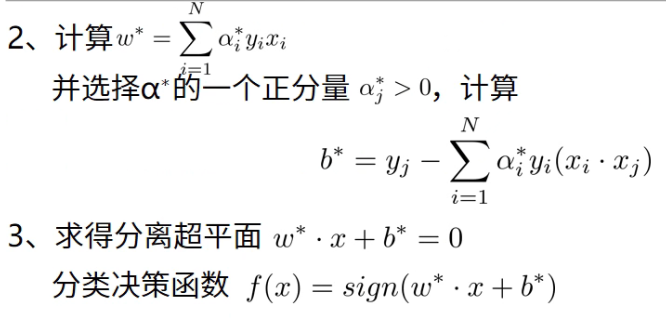
**姓名：李坤璘**

**班级学号：20智能03 2019202216**

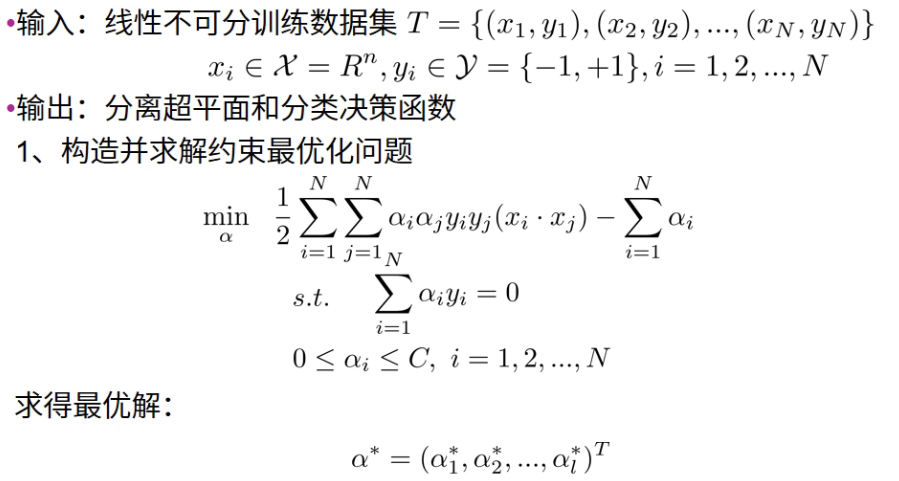
1. **实验目的：**
2. 掌握线性SVM算法的python实现；
3. **实验条件：**
4. PC微机一台和Python环境。
5. **实验原理：**

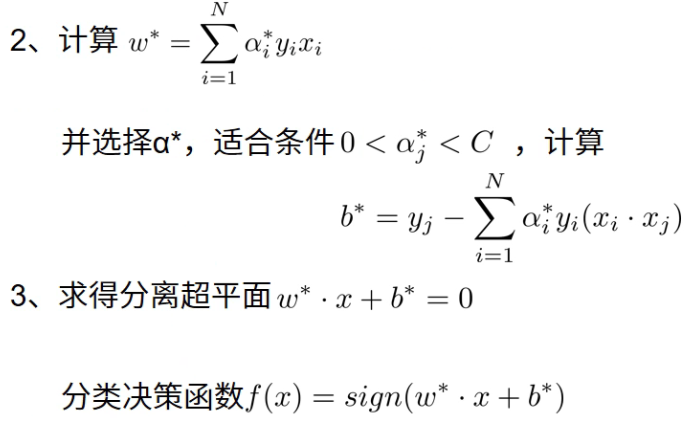
**1、线性可分SVM**





**2、线性不可分线性支持向量机学习算法**

****

****

1. **实验内容：**

1、随机产生两类样本数据：

X=,x1和x2从多元正态分布中随机生成，y={-1,1}

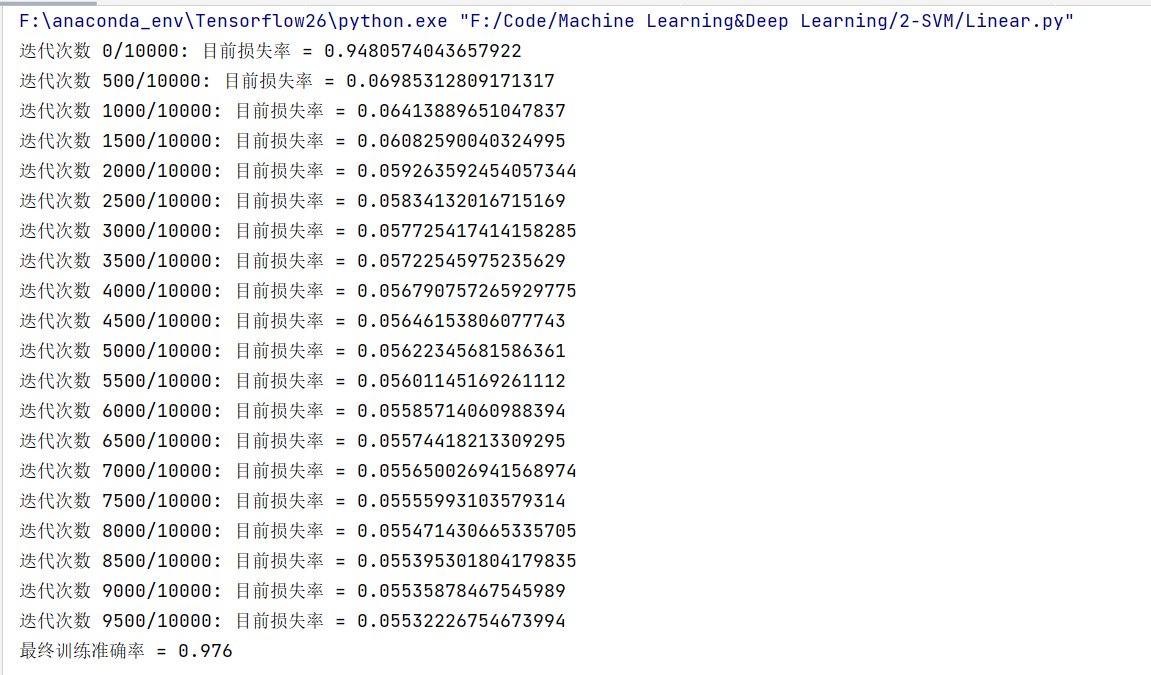
画出样本图。

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)

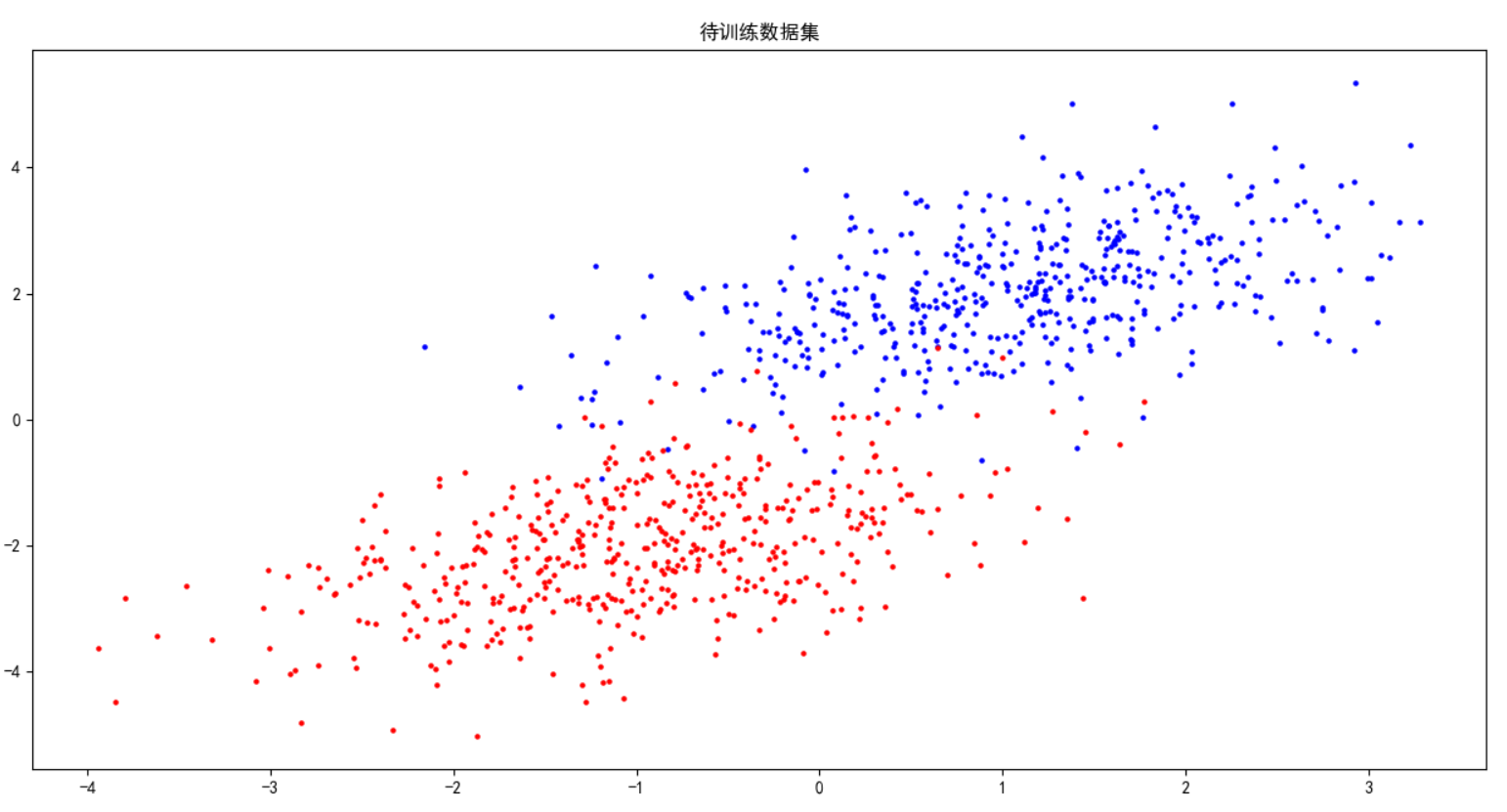
**五、实验代码及结果**

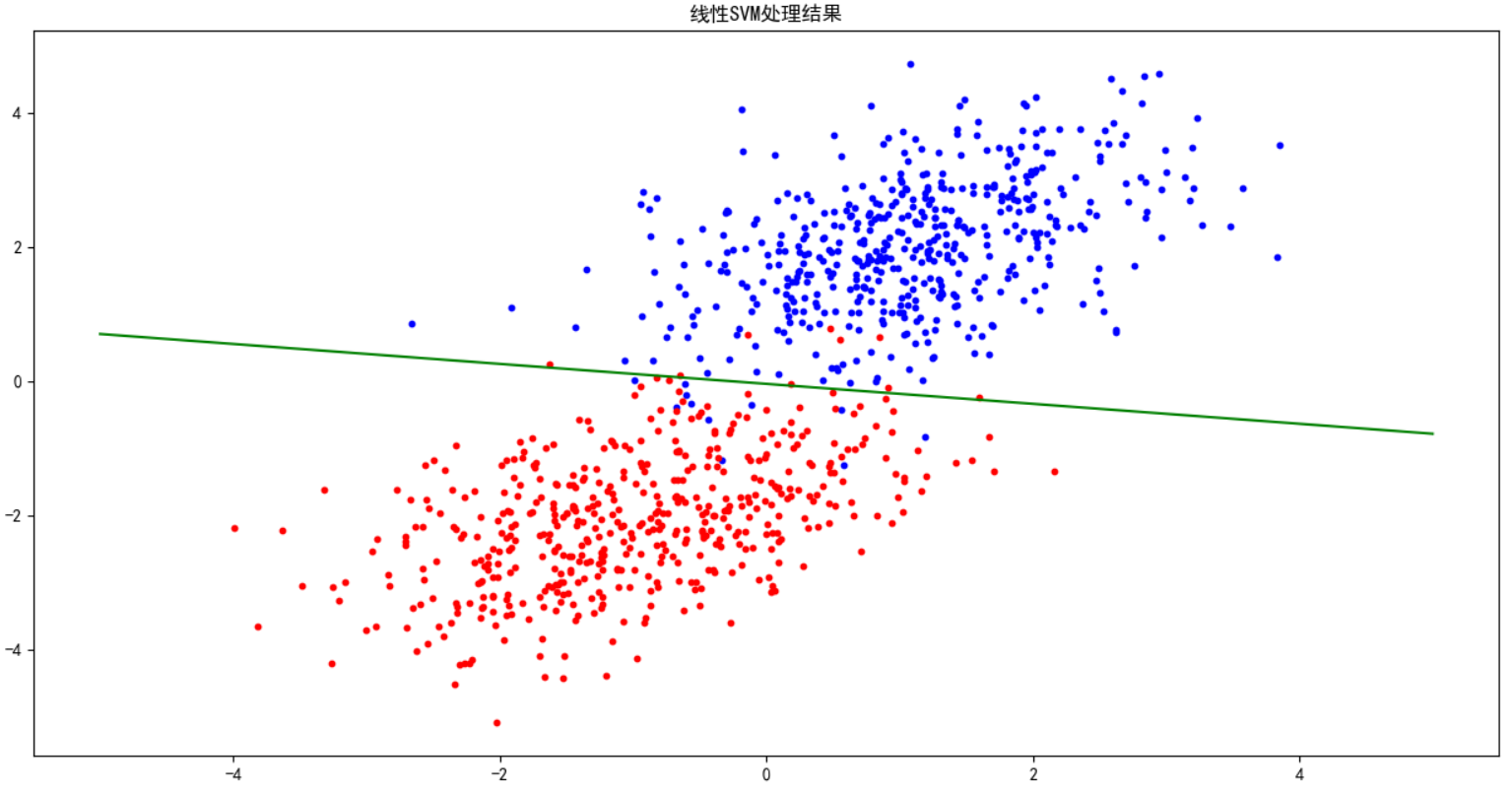
**import numpy as np  
from matplotlib import pyplot as plt  
  
plt.rcParams['font.sans-serif'] = ['SimHei'] *# 用来正常显示中文标签*plt.rcParams['axes.unicode\_minus'] = False *# 用来正常显示负号*'''1.随机生成两类样本数据'''  
mean1 = np.array([1, 2]) *# 样本数据的平均值*cov1 = np.array([[1, 0.5], [0.5, 1]]) *# 协方差矩阵*mean2 = np.array([-1, -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***



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。