**实验三 使用python实现非线性SVM算法**

**姓名：李坤璘**

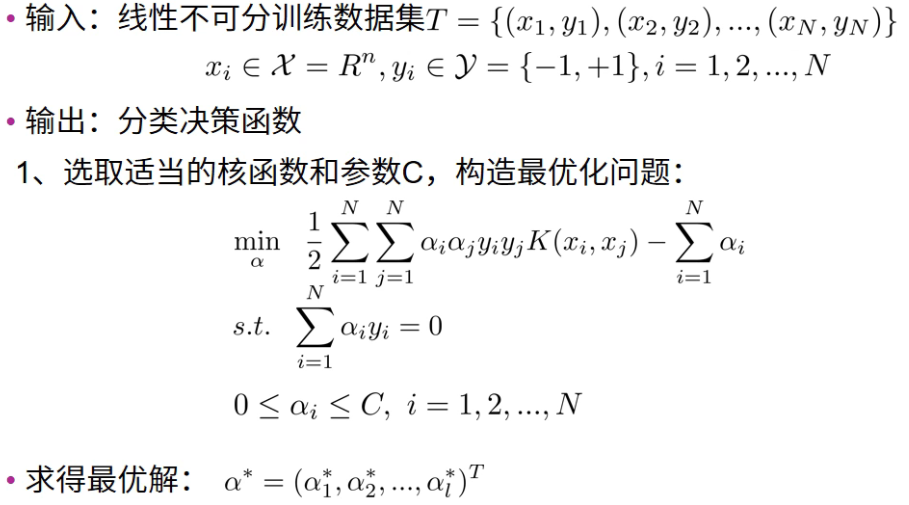
**班级学号：20智能03 2019202216**

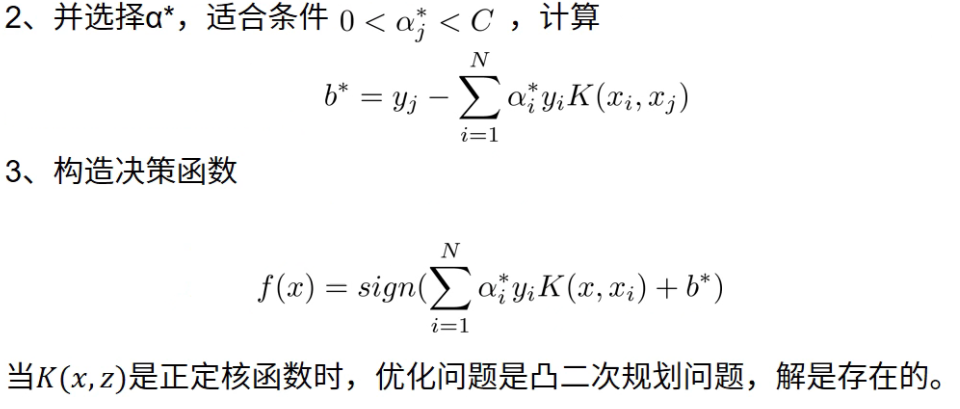
1. **实验目的：**

.掌握非线性SVM算法的python实现。

1. **实验条件：**
2. PC微机一台和Python环境。
3. **实验原理：**

**非线性支持向量机学习算法**





1. **实验内容：**

产生两类均值向量、协方差矩阵如下的样本数据:

(1) 每类产生500个样本作为训练样本;每类产生100个样本作为测试样本;并随机进行标注

(2) 分别画出训练样本和测试样本的分布图;

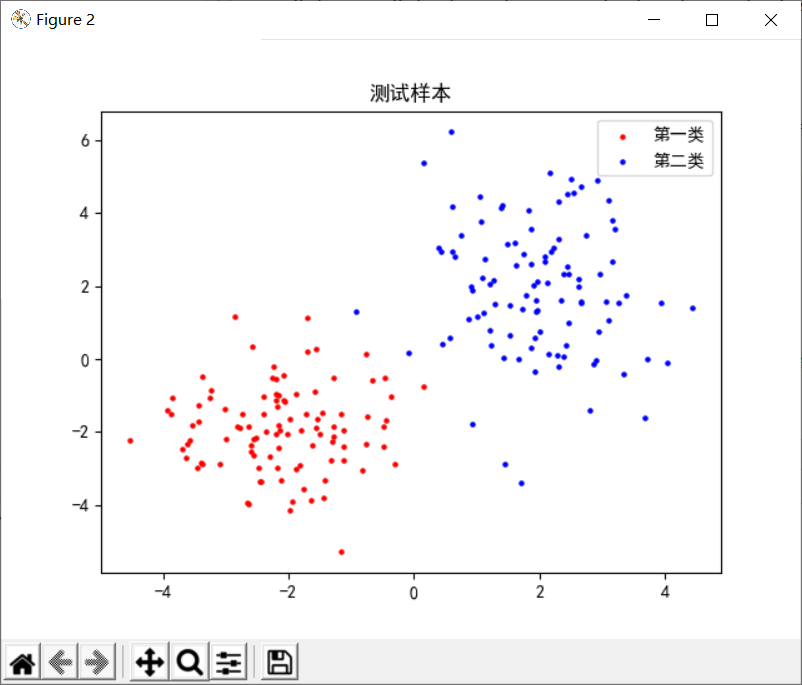
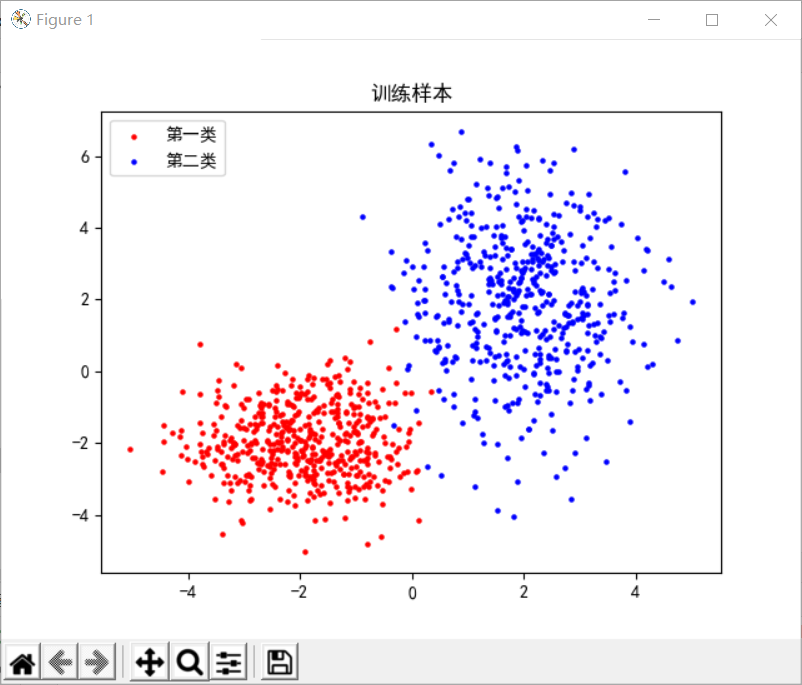
(3) 按最近邻法用训练样本对测试样本分类,计算平均错误率;

(4) 按SVM方法用训练样本对测试样本分类,计算平均错误率;

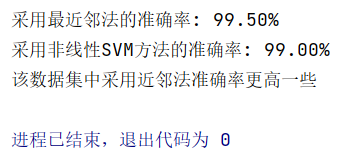
(5) 对两种方法进行对比

**五、实验代码及结果**

**import numpy as np  
from matplotlib import pyplot as plt  
from sklearn.neighbors import KNeighborsClassifier *# k近邻法*from sklearn.svm import SVC *# SVM分类器类*from sklearn.metrics import accuracy\_score  
  
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.产生两类均值向量、协方差矩阵如下的样本数据'''  
mean1, mean2 = [-2, -2], [2, 2]  
cov1, cov2 = [[1, 0], [0, 1]], [[1, 0], [0, 4]]  
  
'''2.每类产生500个样本作为训练样本;每类产生100个样本作为测试样本;并随机进行标注'''  
*# 生成训练样本和测试样本（多元随机正态分布）*train1 = np.random.multivariate\_normal(mean1, cov1, 500)  
train2 = np.random.multivariate\_normal(mean2, cov2, 500)  
test1 = np.random.multivariate\_normal(mean1, cov1, 100)  
test2 = np.random.multivariate\_normal(mean2, cov2, 100)  
*# 合并train和test*train\_x = np.concatenate((train1, train2))  
test\_x = np.concatenate((test1, test2))  
*# 标注样本类别，类别1表示第一类样本，类别-1表示第二类样本*train\_y = np.array([1] \* 500 + [-1] \* 500)  
test\_y = np.array([1] \* 100 + [-1] \* 100)  
  
'''3.画出训练样本和测试样本的分布图'''  
plt.figure(1)  
plt.scatter(train1[:, 0], train1[:, 1], c='r', label='第一类',s=5)  
plt.scatter(train2[:, 0], train2[:, 1], c='b', label='第二类',s=5)  
plt.legend()  
plt.title('训练样本')  
plt.figure(2)  
plt.scatter(test1[:, 0], test1[:, 1], c='r', label='第一类',s=5)  
plt.scatter(test2[:, 0], test2[:, 1], c='b', label='第二类',s=5)  
plt.legend()  
plt.title('测试样本')  
plt.show()  
  
'''4.按最近邻法用训练样本对测试样本分类,计算平均错误率'''  
knn = KNeighborsClassifier(n\_neighbors=1) *# k=1*np.warnings.filterwarnings('ignore')  
knn.fit(train\_x, train\_y)  
test\_predict = knn.predict(test\_x)  
accuracy\_knn = accuracy\_score(test\_y, test\_predict)  
print('采用最近邻法的准确率: {:.2f}%'.format(accuracy\_knn \* 100))  
  
'''5.按SVM方法用训练样本对测试样本分类,计算平均错误率;'''  
*# C：正则化参数，控制对误分类样本的惩罚程度，C越大惩罚越强。  
# kernel：核函数类型  
# gamma：核系数，控制样本映射到高维空间后的分布  
# tol：用于停止训练的误差容忍值*svm = SVC(kernel='rbf',C=1,gamma=0.5,tol=1e-3)  
svm.fit(train\_x, train\_y)  
test\_predict = svm.predict(test\_x)  
accuracy\_svm = accuracy\_score(test\_y, test\_predict)  
print('采用非线性SVM方法的准确率: {:.2f}%'.format(accuracy\_svm \* 100))  
  
'''6.对两种方法进行对比'''  
if accuracy\_svm > accuracy\_knn:  
 print('该数据集中采用SVM准确率更高一些')  
elif accuracy\_svm < accuracy\_knn:  
 print('该数据集中采用近邻法准确率更高一些')  
else:  
 print('该数据集中近邻法和SVM准确率相当')**



训练样本和测试样本散点图如上图所示



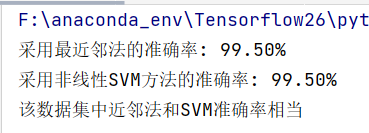
意外发现近邻法的准确率高于svm，分析原因如下：

1.数据集较小：当数据集的规模较小时，近邻法可以更好地对数据进行较为准确的分类。因为SVM在处理小样本数据时，会容易出现过拟合现象，而近邻法则不存在这个问题。

2.数据集分布较为简单：近邻法可以较好地适应数据分布较为简单的情况，对于数据彼此间的关系比较容易识别，近邻法的表现就会比较优越。而SVM更适合处理分布复杂、存在噪声和异常值等情况下的数据。

3.类别之间的边界模糊：当类别之间的边界模糊时，SVM的分类效果可能会不如近邻法，因为SVM采用的是间隔最大化的方法，对于混杂的数据会表现不太好。

再进行多次随机训练得到多组结果如下：



**六、实验总结**

近邻法和SVM是不同的机器学习算法，其适用的场景和模型性质也不同。通常情况下，SVM算法的分类效果要比近邻法好，因为SVM可以比近邻法更好地处理高维数据、非线性分类问题，也更适合处理线性不可分数据。虽然近邻法在特定的情况下可以达到相当高的准确率，但它也有自身的局限性，例如在分类时需要计算大量的相似度距离，而这会在数据量大时带来计算成本的飙升，同时近邻法对于噪声和极端值也比较敏感。因此，当数据集分布复杂、噪声较多的时候，SVM相比于近邻法会有更好的表现，而在数据集较小、分布单纯的情况下，近邻法的准确率可能会更高。

每个算法的参数选择、模型评估等方面都可以对它们的结果产生重要的影响，这也是在应用时需要考虑的因素之一。在具体应用时，应该根据具体数据集的特征和业务场景选择适合的算法和调整合适的参数，以获得更高的准确率。

**七、附录**

实际上我尝试过手写非线性svm，但是由于始终没有将参数调通，所以最终选择学习sklearn进行直接调用，半成品函数如下：

# 近邻法

def nearest\_neighbor(train\_data, train\_labels, test\_data):

    n\_train = len(train\_data)

    n\_test = len(test\_data)

    pred\_labels = np.zeros(n\_test)

    for i in range(n\_test):

        test = test\_data[i]

        min\_dist = np.Inf

        for j in range(n\_train):

            dist = np.linalg.norm(test - train\_data[j])

            if dist < min\_dist:

                min\_dist = dist

                nearest = train\_labels[j]

        pred\_labels[i] = nearest

    return pred\_labels

# 核函数

def rbf\_kernel(x, y, gamma):

    return np.exp(-gamma \* np.linalg.norm(x - y) \*\* 2)

# smo算法

def smo(train\_data, train\_labels, C, toler, gamma, max\_iter):

    n\_train = len(train\_data)

    alphas = np.zeros(n\_train)

    b = 0

    iter = 0

    while iter < max\_iter:

        alpha\_pairs\_changed = 0

        for i in range(n\_train):

            f\_i = np.sum(alphas \* train\_labels \* rbf\_kernel(train\_data, train\_data[i], gamma)) + b

            E\_i = f\_i - train\_labels[i]

            if (train\_labels[i] \* E\_i < -toler and alphas[i] < C) or (train\_labels[i] \* E\_i > toler and alphas[i] > 0):

                j = np.random.choice([k for k in range(n\_train) if k != i])

                f\_j = np.sum(alphas \* train\_labels \* rbf\_kernel(train\_data, train\_data[j], gamma)) + b

                E\_j = f\_j - train\_labels[j]

                alpha\_i\_old, alpha\_j\_old = alphas[i], alphas[j]

                if train\_labels[i] != train\_labels[j]:

                    L = max(0, alphas[j] - alphas[i])

                    H = min(C, C + alphas[j] - alphas[i])

                else:

                    L = max(0, alphas[i] + alphas[j] - C)

                    H = min(C, alphas[i] + alphas[j])

                if L == H:

                    continue

                eta = 2 \* rbf\_kernel(train\_data[i], train\_data[j], gamma) - rbf\_kernel(train\_data[i], train\_data[i], gamma) - rbf\_kernel(train\_data[j], train\_data[j], gamma)

                if eta >= 0:

                    continue

                alphas[j] -= train\_labels[j] \* (E\_i - E\_j) / eta

                alphas[j] = np.clip(alphas[j], L, H)

                if abs(alphas[j] - alpha\_j\_old) < 1e-5:

                    continue

                alphas[i] += train\_labels[i] \* train\_labels[j] \* (alpha\_j\_old - alphas[j])

                b1 = b - E\_i - train\_labels[i] \* (alphas[i] - alpha\_i\_old) \* rbf\_kernel(train\_data[i], train\_data[i], gamma) - train\_labels[j] \* (alphas[j] - alpha\_j\_old) \* rbf\_kernel(train\_data[i], train\_data[j], gamma)

                b2 = b - E\_j - train\_labels[i] \* (alphas[i] - alpha\_i\_old) \* rbf\_kernel(train\_data[i], train\_data[j], gamma) - train\_labels[j] \* (alphas[j] - alpha\_j\_old) \* rbf\_kernel(train\_data[j], train\_data[j], gamma)

                if alphas[i] > 0 and alphas[i] < C:

                    b = b1

                elif alphas[j] > 0 and alphas[j] < C:

                    b = b2

                else:

                    b = (b1 + b2) / 2

                alpha\_pairs\_changed += 1

        if alpha\_pairs\_changed == 0:

            iter += 1

        else:

            iter = 0

    return alphas, b

# SVM

def svm(train\_data, train\_labels, test\_data, C, toler, gamma, max\_iter):

    alphas, b = smo(train\_data, train\_labels, C, toler, gamma, max\_iter)

    n\_train = len(train\_data)

    n\_test = len(test\_data)

    pred\_labels = np.zeros(n\_test)

    for i in range(n\_test):

        f\_i = np.sum(alphas \* train\_labels \* rbf\_kernel(train\_data, test\_data[i], gamma)) + b

        pred\_labels[i] = 1 if f\_i > 0 else 0

    return pred\_labels