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

一、 实验目的：

.掌握非线性 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：

非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：分类决策函数

1、选取适当的核函数和参数C，构造最优化问题：

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

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

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

- 求得最优解：　$\alpha^*=(\alpha_1^*,\alpha_2^*,...,\alpha_l^*)^T$

2、并选择α*，适合条件 $0<\alpha_j^*<C$ ，计算

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

3、构造决策函数

$$f(x)=sign(\sum_{i=1}^{N}\alpha_i^* y_i K(x,x_i)+b^*)$$

当$K(x,z)$是正定核函数时，优化问题是凸二次规划问题，解是存在的。

四、 实验内容：

产生两类均值向量、协方差矩阵如下的样本数据:

$\mu_1 = [-2 \quad -2]$ $\Sigma_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

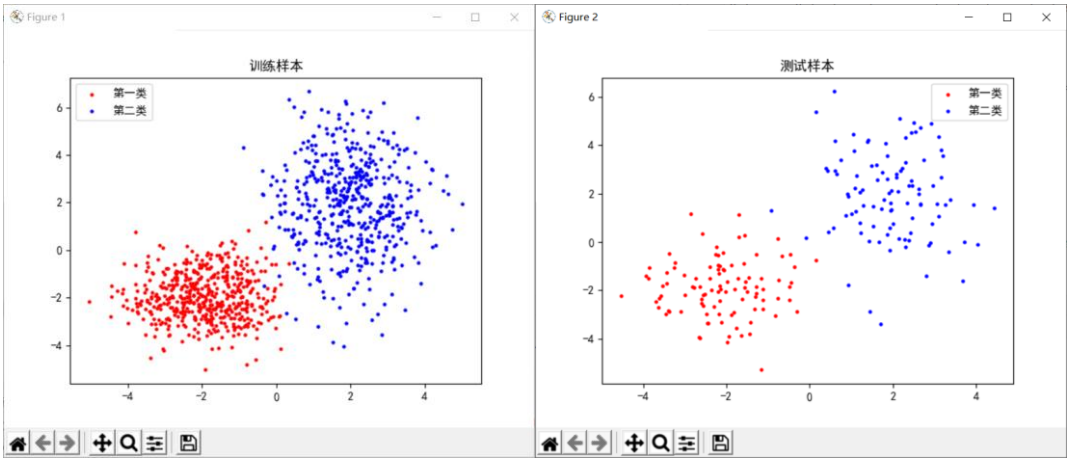
$\mu_2 = [2 \quad 2]$ $\Sigma_2 = \begin{bmatrix} 1 & 0 \\ 0 & 4 \end{bmatrix}$

(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

```
采用最近邻法的准确率： 99.50%
采用非线性SVM方法的准确率： 99.00%
该数据集中采用近邻法准确率更高一些

进程已结束，退出代码为 0
```

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：

```
F:\anaconda_env\Tensorflow26\python.exe
采用最近邻法的准确率： 100.00%
采用非线性SVM方法的准确率： 100.00%
该数据集中近邻法和SVM准确率相当
```

```
F:\anaconda_env\Tensorflow26\pyt
采用最近邻法的准确率： 99.50%
采用非线性SVM方法的准确率： 99.50%
该数据集中近邻法和SVM准确率相当
```

```
F:\anaconda_env\Tensorflow26\python.exe
采用最近邻法的准确率： 99.50%
采用非线性SVM方法的准确率： 100.00%
该数据集中采用SVM准确率更高一些
```

```
F:\anaconda_env\Tensorflow26\python.e
采用最近邻法的准确率： 98.00%
采用非线性SVM方法的准确率： 99.50%
该数据集中采用SVM准确率更高一些
```

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
```