第 3 次实验（本研）基于 SVM 分类实验

- 目标：本实验旨在理解和掌握支持向量机（SVM）分类原理，通过随机生成二维数据点进行分类来理解 SVM 的原理。
- 主要内容：尝试调整不同的 SVM 参数，观察不同参数对分类结果的影响，并与 SVM 分类算法的结果进行比较。
- 重点：核函数的选择、SVM 分类建模

4.6.1. 实验目的

本实验旨在理解和掌握支持向量机（SVM）分类原理。

4.6.2. 实验内容

通过随机生成二维数据点进行分类来理解 SVM 的原理。在实验过程中随机生成若干点来模拟数据，通过对数据的分类来学习 SVM；理解并掌握其他主流分类算法的原理与实现，并与 SVM 分类算法的结果进行比较。

4.6.3. 实验环境

软件：python3

4.6.4. 实验原理

SVM(Support Vector Machine)指的是支持向量机，是常见的一种判别方法。在机器学习领域，是一个有监督的学习模型，通常用来进行模式识别、分类以及回归分析。通俗来讲，它是一种二类分类模型，其基本模型定义为特征空间上的间隔最大的线性分类器。图 1 为线性分类的例子。用一个超平面将不同类别的数据分开。

支持向量机是一种分类算法。

支持向量：支持或支撑平面上把两类类别划分开来的超平面的向量点。

机：一个算法

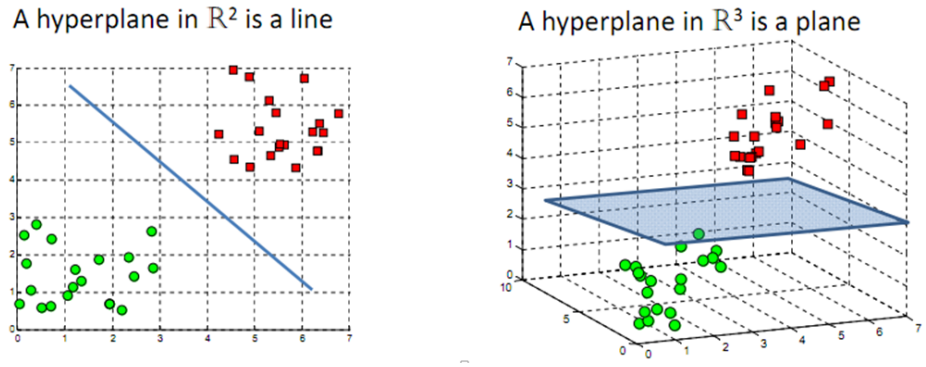


图 1 二维和三维空间上的超平面

支持向量机（Support Vector Machine, SVM）的基本模型是在特征空间上找到最佳的分离超平面使得训练集上正负样本间隔最大。SVM 是用来解决二分类问题的有监督学习算法，在引

入了核方法之后 SVM 也可以用来解决非线性问题。

一般 SVM 有下面三种：

硬间隔支持向量机（线性可分支持向量机）：当训练数据线性可分时，可通过硬间隔最大化学得一个线性可分支持向量机。

软间隔支持向量机：当训练数据近似线性可分时，可通过软间隔最大化学得一个线性支持向量机。

非线性支持向量机：当训练数据线性不可分时，可通过核方法以及软间隔最大化学得一个非线性支持向量机。

（1）硬间隔支持向量机

给定训练样本集$D=\{(\overrightarrow{x_1},y_1),(\overrightarrow{x_2},y_2),\ldots,(\overrightarrow{x_n},y_n),\},y_i\in\{+1,-1\}$，i 表示第 i 个样本，n 表示样本容量。分类学习最基本的想法就是基于训练集 D 在特征空间中找到一个最佳划分超平面将正负样本分开，而 SVM 算法解决的就是如何找到最佳超平面的问题。超平面可通过如下的线性方程来描述：

$$\vec{w}^T\vec{x}+b=0 \tag{1}$$

其中$\vec{w}$表示法向量，决定了超平面的方向；b 表示偏移量，决定了超平面与原点之间的距离。

对于训练数据集 D 假设找到了最佳超平面$\overrightarrow{w^*}\vec{x}+b^*=0$，定义决策分类函数

$$f(\vec{x})=\text{sign}(\overrightarrow{w^*}\vec{x}+b^*)$$

该分类决策函数也称为线性可分支持向量机。

在测试时对于线性可分支持向量机可以用一个样本离划分超平面的距离来表示分类预测的可靠程度，如果样本离划分超平面越远则对该样本的分类越可靠，反之就不那么可靠。

那么，什么样的划分超平面是最佳超平面呢？

对于图 2 有 A、B、C 三个超平面，很明显应该选择超平面 B，也就是说超平面首先应该能满足将两类样本点分开。

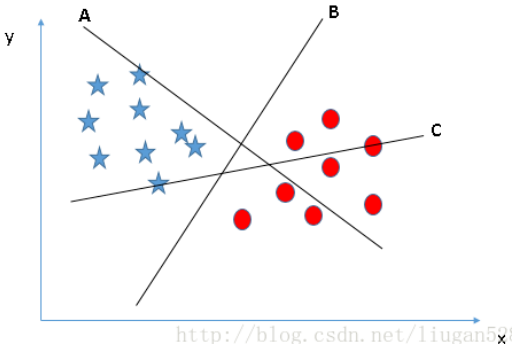


图 2

对于图 3 的 A、B、C 三个超平面，应该选择超平面 C，因为使用超平面 C 进行划分对训练样本局部扰动的“容忍”度最好，分类的鲁棒性最强。例如，由于训练集的局限性或噪声的干

扰，训练集外的样本可能比图 3 中的训练样本更接近两个类目前的分隔界，在分类决策的时候就会出现错误，而超平面 C 受影响最小，也就是说超平面 C 所产生的分类结果是最鲁棒性的、是最可信的，对未见样本的泛化能力最强。

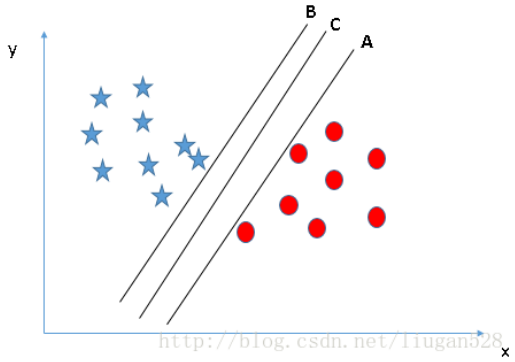


图 3

下面以图 4 中示例进行推导得出最佳超平面。

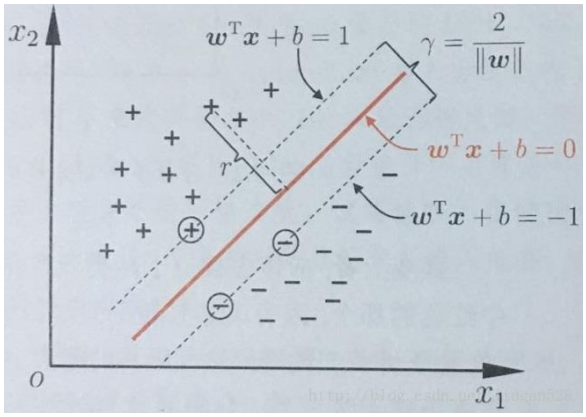


图 4

（2）软间隔支持向量机

在现实任务中很难找到一个超平面将不同类别的样本完全划分开，即很难找到合适的核函数使得训练样本在特征空间中线性可分。退一步说，即使找到了一个可以使训练集在特征空间中完全分开的核函数，也很难确定这个线性可分的结果是不是由于过拟合导致的。解决该问题的办法是在一定程度上运行 SVM 在一些样本上出错，为此引入了“软间隔”（soft margin）的概念，如图 5 所示：

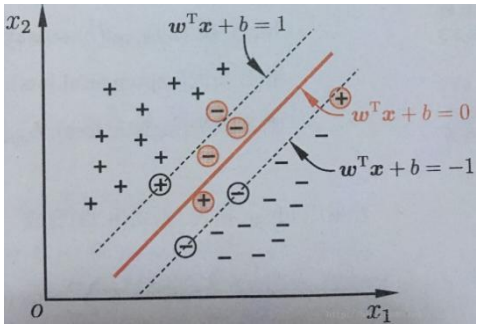


图 5

具体来说，硬间隔支持向量机要求所有的样本均被最佳超平面正确划分，而软间隔支持向量机允许某些样本点不满足间隔大于等于 1 的条件$y_i(\overrightarrow{wx_i}+b)\geq 1$，当然在最大化间隔的时候也要限制不满足间隔大于等于 1 的样本的个数使之尽可能的少。

（3）非线性支持向量机

现实任务中原始的样本空间 D 中很可能并不存在一个能正确划分两类样本的超平面。例如图 5 中所示的问题就无法找到一个超平面将两类样本进行很好的划分。

对于这样的问题可以通过将样本从原始空间映射到特征空间使得样本在映射后的特征空间里线性可分。例如对图 6 做特征映射$z=x^2+y^2$可得到如图 7 所示的样本分布，这样就很好进行线性划分了。

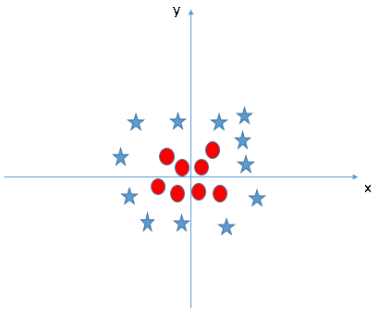


图 6

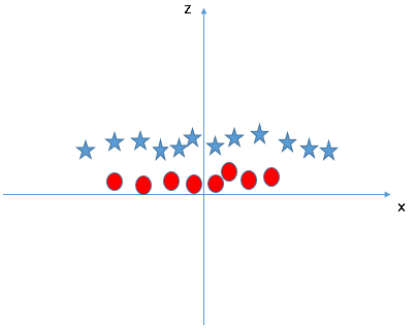


图 7

4.6.5. 实验步骤

本次实验主要学习 SVM 的相关知识，在实验的过程中我们使用随机生成的数据用于 SVM 分类。实验中使用的 SVM 分类器来自 sklearn 包，可以通过

```
from sklearn.svm import SVC
```

进行引用。

实验之前首先看一下 SVC 的定义，对 SVM 有一个初步的印象。

```
def __init__(self, C=1.0, kernel='rbf', degree=3, gamma='auto',
             coef0=0.0, shrinking=True, probability=False,
             tol=1e-3, cache_size=200, class_weight=None,
             verbose=False, max_iter=-1, decision_function_shape='ovr',
             random_state=None):
```

这里我们仅对一些实验相关重要的参数进行介绍：

C：惩罚系数，default = 1.0，同软间隔中的松弛因子

kernel：核函数选择，default = "rbf"，可选 ‘linear’（线性核函数），‘poly’（多项式核函数），‘rbf’（高斯核函数），‘sigmoid’（siigmoid 核函数），

‘precomputed’ degree：多项式核函数 poly 的项数，只有使用 poly 核函数才会调用，其他核函

数会忽略

class_weight：{dict,'balanced'},样本权重，默认均等权重

decision_function_shape：拆分策略，默认 ovr，ovo 一对一 1 策略被舍弃

random_state：随机数生成种子

接下来开始进行实验

1、 引入需要使用的包

```
from sklearn.svm import SVC
import matplotlib.pyplot as plt
import numpy as np
import math
import random
from sklearn.model_selection import train_test_split
```

SVC 是我们使用的 SVM 分类器，plt 用于画图，numpy 是科学计算库，用于矩阵操作，random 用于生成随机数来模拟我们想要的数据。Train_test_split 用于分割数据。Math 用于数学计算

2、 接下来编写生成模拟数据的函数

```
def mock_data(point_count, point_type = 1):
    """
    模拟生成若干个点
    :param point_count:点的个数
    :param point_type: 1表示生成点线性可分，2表示线性不可分
    :return:points, labels
    """
    points = []
    labels = []
    if point_type == 1:
        return [[1,3],[2,2.5],[3.5,1]],[0,0,1]
    elif point_type == 2:
        for i in range(point_count // 2):
            point_x = random.uniform(0,10)
            point_y = random.uniform(point_x+1,10)
            points.append([point_x, point_y])
            labels.append(0)
        for i in range(point_count // 2):
            point_y = random.uniform(0,10)
            point_x = random.uniform(point_y+1,10)
            points.append([point_x, point_y])
            labels.append(1)
        for i in range(point_count // 6):
            point_y = random.uniform(0,10)
            point_x = random.uniform(0,10)
            points.append([point_x, point_y])
            labels.append(random.choice([0,1]))
    elif point_type == 3:
        for i in range(point_count // 2):
            point_x = random.uniform(-2,2)
            point_y = random.uniform(-math.sqrt(4-point_x*point_x),math.sqrt(4-point_x*point_x))
            points.append([point_x,point_y])
            labels.append(0)
        for i in range(point_count // 2):
            point_x = random.uniform(-2, 2)
            point_y = random.choice([random.uniform(-4, -math.sqrt(4-point_x*point_x)), random.uniform(math.sqrt(4-point_x*point_x),4)])
            points.append([point_x,point_y])
            labels.append(1)

    else:
        raise Exception("type类型错误{0}".format(point_type))
    return points,labels
```

我们使用参数 point_type 来决定生成二维数据的类型。point_type =1 时生成的数据在平面

上可分，point_type =2 时数据基本可分，但是有少许噪音，point_type=3 时生成无法线性可分的数据。

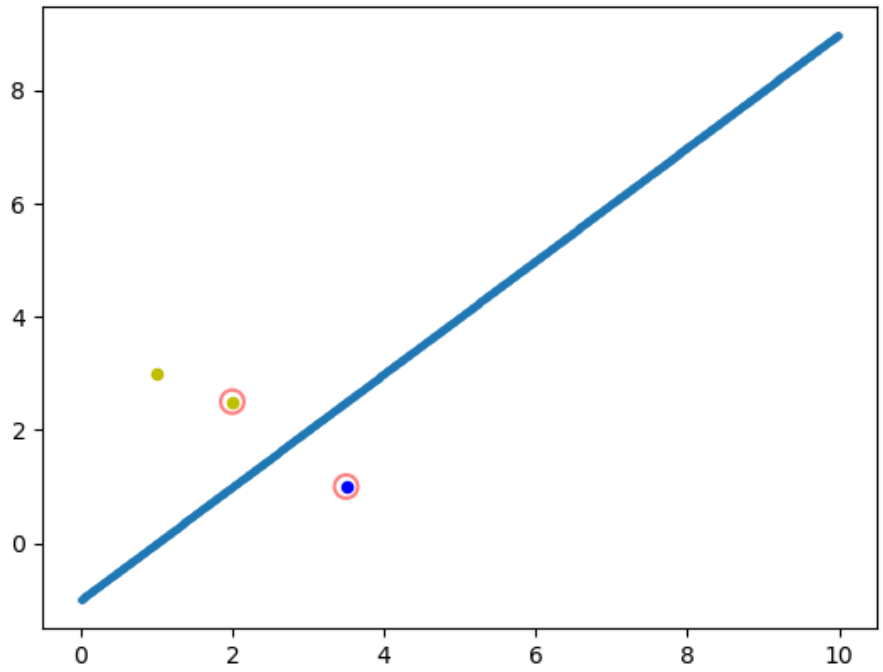
3、 编写函数用于绘制分类结果及超平面

```
def plot_point(dataArr, labelArr, Support_vector_index, W, b):
    for i in range(np.shape(dataArr)[0]):
        if labelArr[i] == 1:
            plt.scatter(dataArr[i][0], dataArr[i][1], c='b', s=20)
        else:
            plt.scatter(dataArr[i][0], dataArr[i][1], c='y', s=20)
    for j in Support_vector_index:
        plt.scatter(dataArr[j][0], dataArr[j][1], s=100, c='', alpha=0.5, linewidth=1.5, edgecolor='red')
    x = np.arange(0, 10, 0.01)
    y = (W[0][0] * x + b) / (-1 * W[0][1])
    plt.scatter(x, y, s=5, marker='h')
    plt.show()
```

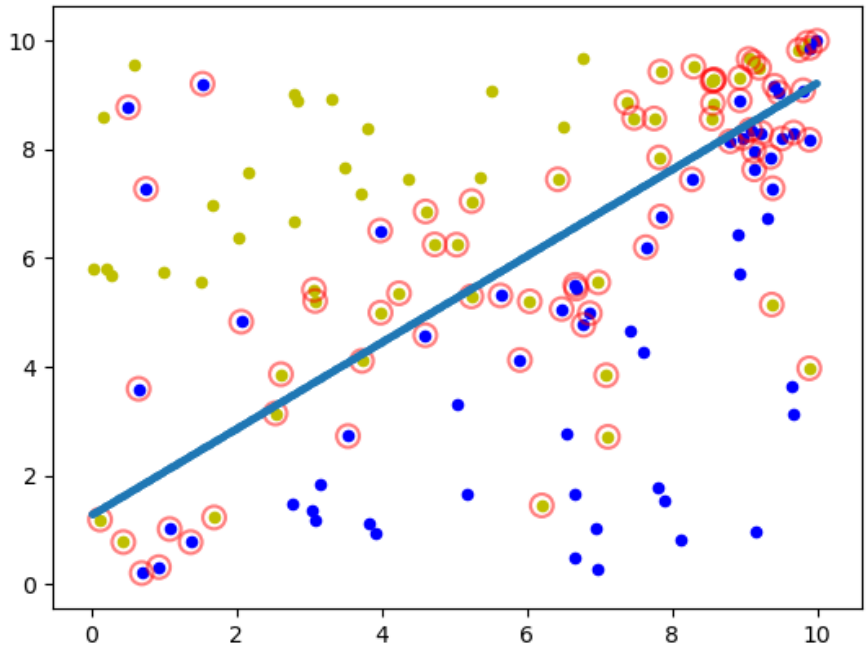
4、 接下来编写主函数，生成第一种类型的数据进行分类并查看效果

```
if __name__ == "__main__":
    # 读取数据,针对二维线性可分数据
    dataArr, labelArr = mock_data(100,point_type=1)
    # 定义SVM分类器
    clf = SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
              decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
              max_iter=-1, probability=False, random_state=None, shrinking=True,
              tol=0.001, verbose=False)
    # fit训练数据
    clf.fit(dataArr, labelArr)
    # 获取模型返回值
    n_Support_vector = clf.n_support_  # 支持向量个数
    Support_vector_index = clf.support_  # 支持向量索引
    W = clf.coef_  # 方向向量W
    b = clf.intercept_  # 截距项b
    # 绘制分类超平面
    plot_point(dataArr, labelArr, Support_vector_index, W, b)
```

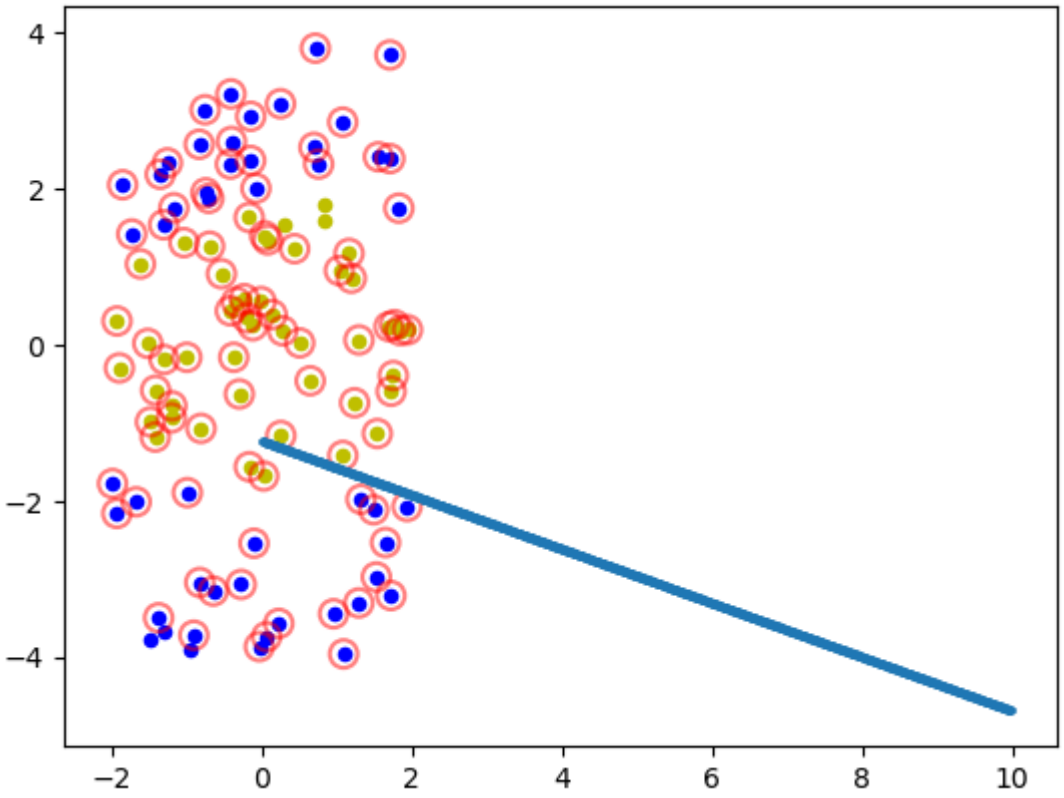
5、 结果如图



6、 再生成第二种数据，其他参数不变进行测试，结果如图



7、 再生成第三种数据，其他参数不变进行测试，结果如图



此时，由于数据无法线性可分，线性分类的 SVM 已经无法很好的进行分类。

8、 初步有了印象之后，我们编写代码来量化 SVM 分类的成功率。首先修改 main 函数，将数据划分为训练集和测试集，然后输出测试结果。

```
if __name__ == "__main__":
    # 读取数据,针对二维线性可分数据
    dataArr, labelArr = mock_data(100,point_type=3)

    if __name__ == "__main__":
        # 读取数据,针对二维线性不可分数据
        dataArr, labelArr = mock_data(100,3)
        for i in range(np.shape(dataArr)[0]):
            if labelArr[i] == 1:
                plt.scatter(dataArr[i][0], dataArr[i][1], c='b', s=20)
            else:
                plt.scatter(dataArr[i][0], dataArr[i][1], c='y', s=20)
        plt.show()
        # 交叉验证划分数据集, train: test = 0.8 : 0.2
        X_train, X_test, y_train, y_test = train_test_split(dataArr, labelArr, test_size=.2, random_state=0)
        # 初始化模型参数
        clf = SVC(cache_size=200, class_weight=None, coef0=0.0, C=1.0,
                  decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
                  max_iter=-1, probability=False, random_state=None, shrinking=True,
                  tol=0.001, verbose=False)
        clf.fit(X_train, y_train)
        # 预测X_test
        predict_list = clf.predict(X_test)
        # 预测精度
        precition = clf.score(X_test, y_test)
        print('precition is : ', precition * 100, "%")
        # 获取模型返回值
        n_Support_vector = clf.n_support_  # 支持向量个数
        print("支持向量个数为： ", n_Support_vector)
        Support_vector_index = clf.support_  # 支持向量索引
        plot_point(dataArr, labelArr, Support_vector_index)
```

9、 修改 plot_point 函数使其可以输出更为维度的结果

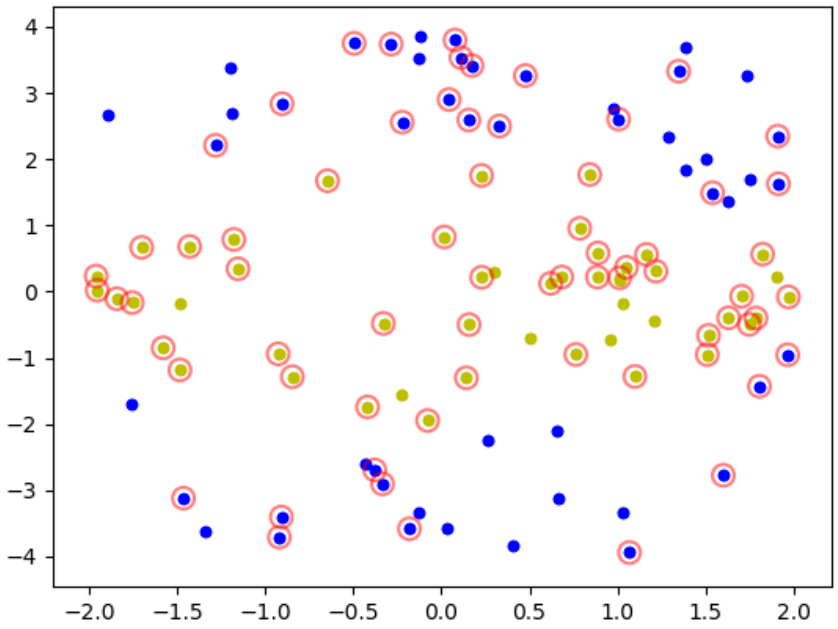
```
def plot_point(dataArr, labelArr, Support_vector_index):
    for i in range(np.shape(dataArr)[0]):
        if labelArr[i] == 1:
            plt.scatter(dataArr[i][0], dataArr[i][1], c='b', s=20)
        else:
            plt.scatter(dataArr[i][0], dataArr[i][1], c='y', s=20)
    for j in Support_vector_index:
        plt.scatter(dataArr[j][0], dataArr[j][1], s=100, c='', alpha=0.5, linewidth=1.5, edgecolor='red')
    plt.show()
```

10、 接下来测试若干次第三组数据的 SVM 分类准确率

```
D:\Program\python3.6.6\python.exe "D:/OneDrive - business-cn/beihang university/YANERXIA/graduate/code/yujiannan.py"
precition is :  65.0 %
支持向量个数为:  [35 35]

Process finished with exit code 0
```

经过观察，若干次实验中分类准确度并不稳定，最高在 80%左右，最低在 20%左右。



11、 此时我们将 SVM 的参数中的核函数切换为 rbf 函数，再测试若干组数据观察成功率

```
# 初始化模型参数
clf = SVC(cache_size=200, class_weight=None, coef0=0.0, C=1.0,
          decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
          max_iter=-1, probability=False, random_state=None, shrinking=True,
          tol=0.001, verbose=False)
clf.fit(X_train, y_train)
```

观察结果，可以看见使用 rbf 核函数之后，准确率大大提升，对于相当一部分的实验都达到了 100%的成功率，由此可见 rbf 核函数对这类数据具有更好的分类效果。

4.6.6. 扩展实验

1）请同学们用 sklearn 中的鸢尾花数据集使用 SVM 库进行分类。在分类的过程中尝试调整其他的 SVM 参数，观察不同的参数对结果的影响并尝试尽可能提高准确率，并对实验结果进行对比分析；

2）用 SVM 实现 MNIST 手写数字图像识别，观察不同的参数对结果的影响并尝试尽可能提高准确率，并对实验结果进行对比分析（数据已经传至平台）。

要求这两扩展实验至少做一个。