第 10 章 支持向量机

内容提要

- 简单介绍支持向量机
- 利用核函数对数据进行空间转换
- 利用 SMO 进行优化
- 将 SVM 和其他分类器进行对比

10.1 基于最大间隔分隔数据

先考虑图10.1方框 A 中的两组数据，它们之间已经分隔得足够开，因此很容易就可以在图中画出一条直线将两组数据点分开。在这种情况下，这组数据被称为线性可分（linearly separable）数据。

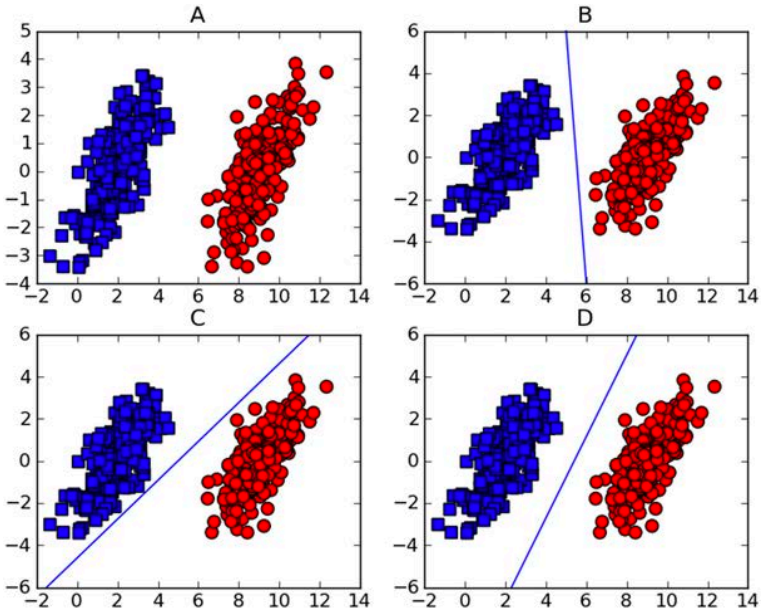

图 10.1: A 框中给出了一个线性可分的数据集，B、C、D 框中各自给出了一条可以将两类数据分开的直线

上述将数据集分隔开来的直线称为分隔超平面（separating hyperplane）。在上面给出的例子中，由于数据点都在二维平面上，所以此时分隔超平面就只是一条直线。但是，如果所给的数据集是三维的，那么此时用来分隔数据的就是一个平面。显而易见，更高维的情况可以依此类推。如果数据集是 1024 维的，那么就需要一个 1023 维的某某对象来对数据进行分隔。这个 1023 维的某某对象到底应该叫什么？$N \times 1$ 维呢？该对象被称为超平面（hyperplane），也就是分类的决策边界。分布在超平面一侧的所有数据都属于某个类别，而分布在另一侧的所有数据则属于另一个类别。

我们希望能采用这种方式来构建分类器，即如果数据点离决策边界越远，那么其最后的预测结果也就越可信。考虑图10.1框 B 到框 D 中的三条直线，它们都能将数据分隔开，但是其中哪一条最好呢？是否应该最小化数据点到分隔超平面的平均距离来求最佳直线？如果是那样，图10.1的 B 和 C 框中的直线是否真的就比 D 框中的直线好呢？如果这样做，是不是有点寻找最佳拟合直线的感觉？是的，上述做法确实有点像直线拟合，但这并非最佳方案。我们

希望找到离分隔超平面最近的点，确保它们离分隔面的距离尽可能远。这里点到分隔面的距离被称为间隔（margin）。我们希望间隔尽可能地大，这是因为如果我们犯错或者在有限数据上训练分类器的话，我们希望分类器尽可能健壮。

支持向量（support vector）就是离分隔超平面最近的那些点。接下来要试着最大化支持向量到分隔面的距离，需要找到此问题的优化求解方法。

10.2 寻找最大间隔

如何求解数据集的最佳分隔直线？先来看看图10.2。分隔超平面的形式可以写成 $w^T x + b$。要计算点 A 到分隔超平面的距离，就必须给出点到分隔面的法线或垂线的长度，该值为 $|w^T A + b|/\|w\|$。这里的常数 b 类似于 Logistic 回归中的截距 w_0。这里的向量 w 和常数 b 一起描述了所给数据的分隔线或超平面。

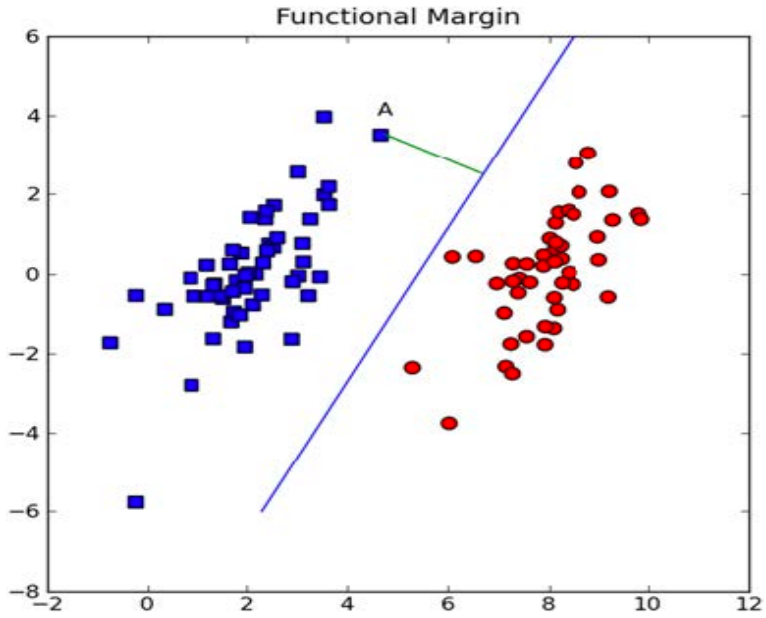

图 10.2: 点 A 到分隔平面的距离就是该点到分隔面的法线长度

10.3 分类器求解的优化问题

支持向量机的优化问题需要大量的背景知识，建议大家查阅相关教材，独立推导相关公式。

支持向量机的最终优化公式为：

$$
\begin{aligned}
&max_{\alpha}[\sum_{i=1}^{m}\alpha - \frac{1}{2}\sum_{i,j=1}^{m} label^{(i)} label^{(j)} \alpha_i \alpha_j < x^{(i)}, x^{(j)} >] \\
&\alpha \geq 0 \ \ and \ \ \sum_{i=1}^{m} \alpha_i label^{(i)} = 0
\end{aligned}
\tag{10.1}
$$

这里的常数 C 用于控制“最大化间隔”和“保证大部分点的函数间隔小于 1.0”这两个目标的

权重。

10.4 SMO 高效优化算法

SMO 算法序列最小优化（Sequential Minimal Optimization）是将大优化问题分解为多个小优化问题来求解的。这些小优化问题往往很容易求解，并且对它们进行顺序求解的结果与将它们作为整体来求解的结果是完全一致的。在结果完全相同的同时，SMO 算法的求解时间短很多。

SMO 算法的目标是求出一系列 α 和 b，一旦求出了这些 α，就很容易计算出权重向量 w 并得到分隔超平面。

SMO 算法的工作原理是：每次循环中选择两个 α 进行优化处理。一旦找到一对合适的 α，那么就增大其中一个同时减小另一个。这里所谓的“合适”就是指两个 alpha 必须要符合一定的条件，条件之一就是这两个 α 必须要在间隔边界之外，而其第二个条件则是这两个 α 还没有进行过区间化处理或者不在边界上。

Step 1: 加载数据

```
from time import sleep
import matplotlib.pyplot as plt
import numpy as np
import random
import types

"""
函数说明:读取数据

Parameters:
    fileName - 文件名
Returns:
    dataMat - 数据矩阵
    labelMat - 数据标签
"""
def loadDataSet(fileName):
    dataMat = []; labelMat = []
    fr = open(fileName)
    for line in fr.readlines():                                    #逐行读取，滤除空格等
        lineArr = line.strip().split('\t')
        dataMat.append([float(lineArr[0]), float(lineArr[1])]) #添加数据
        labelMat.append(float(lineArr[2]))                         #添加标签
    return dataMat,labelMat
```

Step 2: SMO 算法

SMO 算法中的辅助函数

```
"""
函数说明:随机选择alpha

Parameters:
```

```
    i - alpha
    m - alpha参数个数
Returns:
    j
"""
def selectJrand(i, m):
    j = i                                #选择一个不等于i的j
    while (j == i):
        j = int(random.uniform(0, m))
    return j

"""
函数说明:修剪alpha

Parameters:
    aj - alpha值
    H - alpha上限
    L - alpha下限
Returns:
    aj - alpah值
"""
def clipAlpha(aj,H,L):
    if aj > H:
        aj = H
    if L > aj:
        aj = L
    return aj
```

简化版 SMO 算法:

```
 """
函数说明:简化版SMO算法

Parameters:
    dataMatIn - 数据矩阵
    classLabels - 数据标签
    C - 松弛变量
    toler - 容错率
    maxIter - 最大迭代次数
Returns:
    无
"""
def smoSimple(dataMatIn, classLabels, C, toler, maxIter):
    #转换为numpy的mat存储
    dataMatrix = np.mat(dataMatIn); labelMat = np.mat(classLabels).transpose()
    #初始化b参数，统计dataMatrix的维度
    b = 0; m,n = np.shape(dataMatrix)
    #初始化alpha参数，设为0
    alphas = np.mat(np.zeros((m,1)))
    #初始化迭代次数
```

```
iter_num = 0
#最多迭代matIter次
while (iter_num < maxIter):
    alphaPairsChanged = 0
    for i in range(m):
        #步骤1：计算误差Ei
        fXi = float(np.multiply(alphas,labelMat).T*(dataMatrix*dataMatrix[i,:].T)) + b
        Ei = fXi - float(labelMat[i])
        #优化alpha，更设定一定的容错率。
        if ((labelMat[i]*Ei < -toler) and (alphas[i] < C)) or ((labelMat[i]*Ei > toler) and (alphas
            [i] > 0)):
            #随机选择另一个与alpha_i成对优化的alpha_j
            j = selectJrand(i,m)
            #步骤1：计算误差Ej
            fXj = float(np.multiply(alphas,labelMat).T*(dataMatrix*dataMatrix[j,:].T)) + b
            Ej = fXj - float(labelMat[j])
            #保存更新前的aplpha值，使用深拷贝
            alphaIold = alphas[i].copy(); alphaJold = alphas[j].copy();
            #步骤2：计算上下界L和H
            if (labelMat[i] != labelMat[j]):
                L = max(0, alphas[j] - alphas[i])
                H = min(C, C + alphas[j] - alphas[i])
            else:
                L = max(0, alphas[j] + alphas[i] - C)
                H = min(C, alphas[j] + alphas[i])
            if L==H: print("L==H"); continue
            #步骤3：计算eta
            eta = 2.0 * dataMatrix[i,:]*dataMatrix[j,:].T - dataMatrix[i,:]*dataMatrix[i,:].T -
                dataMatrix[j,:]*dataMatrix[j,:].T
            if eta >= 0: print("eta>=0"); continue
            #步骤4：更新alpha_j
            alphas[j] -= labelMat[j]*(Ei - Ej)/eta
            #步骤5：修剪alpha_j
            alphas[j] = clipAlpha(alphas[j],H,L)
            if (abs(alphas[j] - alphaJold) < 0.00001): print("alpha_j变化太小"); continue
            #步骤6：更新alpha_i
            alphas[i] += labelMat[j]*labelMat[i]*(alphaJold - alphas[j])
            #步骤7：更新b_1和b_2
            b1 = b - Ei- labelMat[i]*(alphas[i]-alphaIold)*dataMatrix[i,:]*dataMatrix[i,:].T -
                labelMat[j]*(alphas[j]-alphaJold)*dataMatrix[i,:]*dataMatrix[j,:].T
            b2 = b - Ej- labelMat[i]*(alphas[i]-alphaIold)*dataMatrix[i,:]*dataMatrix[j,:].T -
                labelMat[j]*(alphas[j]-alphaJold)*dataMatrix[j,:]*dataMatrix[j,:].T
            #步骤8：根据b_1和b_2更新b
            if (0 < alphas[i]) and (C > alphas[i]): b = b1
            elif (0 < alphas[j]) and (C > alphas[j]): b = b2
            else: b = (b1 + b2)/2.0
            #统计优化次数
            alphaPairsChanged += 1
            #打印统计信息
```

```
                print("第%d次迭代 样本:%d, alpha优化次数:%d" % (iter_num,i,alphaPairsChanged))
        #更新迭代次数
        if (alphaPairsChanged == 0): iter_num += 1
        else: iter_num = 0
        print("迭代次数: %d" % iter_num)
    return b,alphas
```

Step 3: 分类结果可视化

```
"""
函数说明:计算w

Parameters:
    dataMat - 数据矩阵
    labelMat - 数据标签
    alphas - alphas值
Returns:
    无
"""
def get_w(dataMat, labelMat, alphas):
    alphas, dataMat, labelMat = np.array(alphas), np.array(dataMat), np.array(labelMat)
    w = np.dot((np.tile(labelMat.reshape(1, -1).T, (1, 2)) * dataMat).T, alphas)
    return w.tolist()

"""
函数说明:分类结果可视化

Parameters:
    dataMat - 数据矩阵
    w - 直线法向量
    b - 直线解决
Returns:
    无
"""
def showClassifer(dataMat, w, b):
    #绘制样本点
    data_plus = []                                  #正样本
    data_minus = []                                 #负样本
    for i in range(len(dataMat)):
        if labelMat[i] > 0:
            data_plus.append(dataMat[i])
        else:
            data_minus.append(dataMat[i])
    data_plus_np = np.array(data_plus)              #转换为numpy矩阵
    data_minus_np = np.array(data_minus)            #转换为numpy矩阵
    plt.scatter(np.transpose(data_plus_np)[0], np.transpose(data_plus_np)[1], s=30, alpha=0.7) #正样本
        散点图
    plt.scatter(np.transpose(data_minus_np)[0], np.transpose(data_minus_np)[1], s=30, alpha=0.7) #负样
        本散点图
    #绘制直线
```

```
x1 = max(dataMat)[0]
x2 = min(dataMat)[0]
a1, a2 = w
b = float(b)
a1 = float(a1[0])
a2 = float(a2[0])
y1, y2 = (-b- a1*x1)/a2, (-b - a1*x2)/a2
plt.plot([x1, x2], [y1, y2])
#找出支持向量点
for i, alpha in enumerate(alphas):
    if abs(alpha) > 0:
        x, y = dataMat[i]
        plt.scatter([x], [y], s=150, c='none', alpha=0.7, linewidth=1.5, edgecolor='red')
plt.show()
```

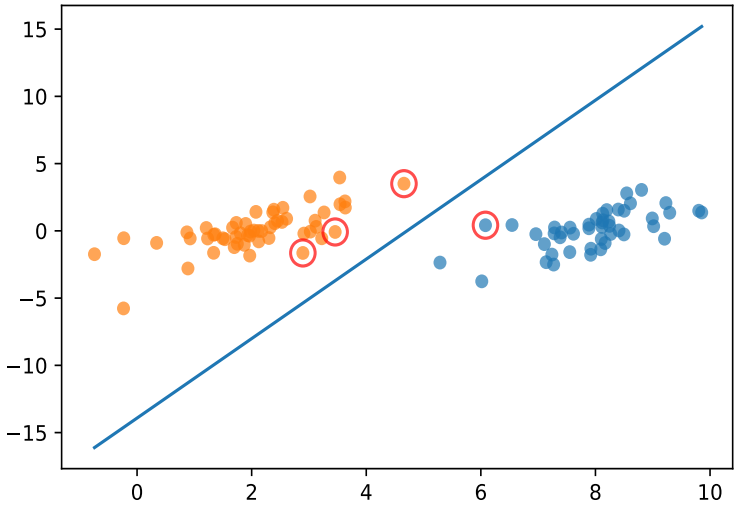

图 10.3: 示例数据集上运行简化版 SMO 算法后得到的结果，包括画圈的支持向量与分隔超平面

10.5 在复杂数据上应用核函数

针对非线性可分的数据，支持向量机很难对数据进行建模。针对非线性可分的数据，如10.4，我们能否像线性情况一样，利用强大的工具来捕捉数据中的这种模式？显然，答案是肯定的。接下来，我们就要使用一种称为核函数（kernel）的工具将数据转换成易于分类器理解的形式。

有大量关于核函数的资料，请自行查阅。

10.5.1 径向基核函数

径向基函数是 SVM 中常用的一个核函数。径向基函数是一个采用向量作为自变量的函数，能够基于向量距离运算输出一个标量。这个距离可以是从 <0,0> 向量或者其他向量开始计算的距离。接下来，我们将会使用到径向基函数的高斯版本，其具体公式为：

$$k(x,y) = exp(-\frac{\|x-y\|^2}{2\sigma^2}) \tag{10.2}$$

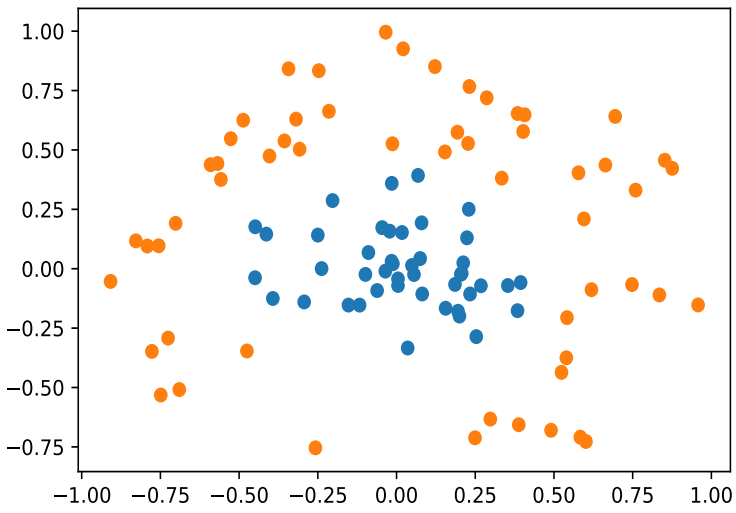

图 10.4: 这个数据在二维平面中很难用一条直线分隔。

其中，σ 是用户定义的用于确定到达率（reach）或者说函数值跌落到 0 的速度参数。

上述高斯核函数将数据从其特征空间映射到更高维的空间，具体来说这里是映射到一个无穷维的空间。

核转换函数：

```
class optStruct:
    """
    数据结构，维护所有需要操作的值
    Parameters:
        dataMatIn - 数据矩阵
        classLabels - 数据标签
        C - 松弛变量
        toler - 容错率
        kTup - 包含核函数信息的元组,第一个参数存放核函数类别，第二个参数存放必要的核函数需要用到的参数
    """
    def __init__(self, dataMatIn, classLabels, C, toler, kTup):
        self.X = dataMatIn                                  #数据矩阵
        self.labelMat = classLabels                         #数据标签
        self.C = C                                          #松弛变量
        self.tol = toler                                    #容错率
        self.m = np.shape(dataMatIn)[0]                     #数据矩阵行数
        self.alphas = np.mat(np.zeros((self.m,1)))          #根据矩阵行数初始化alpha参数为0
        self.b = 0                                          #初始化b参数为0
        self.eCache = np.mat(np.zeros((self.m,2)))          #根据矩阵行数初始化虎误差缓存，第一列为是否有效的标
            志位，第二列为实际的误差E的值。
        self.K = np.mat(np.zeros((self.m,self.m)))          #初始化核K
        for i in range(self.m):                             #计算所有数据的核K
            self.K[:,i] = kernelTrans(self.X, self.X[i,:], kTup)
```

```
def kernelTrans(X, A, kTup):
    """
    通过核函数将数据转换更高维的空间
    Parameters:
        X - 数据矩阵
        A - 单个数据的向量
        kTup - 包含核函数信息的元组
    Returns:
        K - 计算的核K
    """
    m,n = np.shape(X)
    K = np.mat(np.zeros((m,1)))
    if kTup[0] == 'lin': K = X * A.T                    #线性核函数,只进行内积。
    elif kTup[0] == 'rbf':                              #高斯核函数,根据高斯核函数公式进行计算
        for j in range(m):
            deltaRow = X[j,:] - A
            K[j] = deltaRow*deltaRow.T
        K = np.exp(K/(-1*kTup[1]**2))                   #计算高斯核K
    else: raise NameError('核函数无法识别')
    return K                                            #返回计算的核K
```

kTup 是一个包含核函数信息的元组，待会儿我们就能看到它的作用了。在初始化方法结束时，矩阵 K 先被构建，然后再通过调用函数 kernelTrans() 进行填充。全局的 K 值只需计算一次。然后，当想要使用核函数时，就可以对它进行调用。这也省去了很多冗余的计算开销。

当计算矩阵 K 时，该过程多次调用了函数 kernelTrans()。该函数有 3 个输入参数：2 个数值型变量和 1 个元组。元组 kTup 给出的是核函数的信息。元组的第一个参数是描述所用核函数类型的一个字符串，其他 2 个参数则都是核函数可能需要的可选参数。该函数首先构建出了一个列向量，然后检查元组以确定核函数的类型。这里只给出了 2 种选择，但是依然可以很容易地通过添加 elif 语句来扩展到更多选项。

在线性核函数的情况下，内积计算在“所有数据集”和“数据集中的一行”这两个输入之间展开。在径向基核函数的情况下，在 for 循环中对于矩阵的每个元素计算高斯函数的值。而在 for 循环结束之后，我们将计算过程应用到整个向量上去。

最后，如果遇到一个无法识别的元组，程序就会抛出异常，因为在这种情况下不希望程序再继续运行，这一点相当重要。

使用核函数时需要对 innerL() 及 calcEk() 函数进行的修改：

```
ef innerL(i, oS):
    """
    优化的SMO算法
    Parameters:
        i - 标号为i的数据的索引值
        oS - 数据结构
    Returns:
        1 - 有任意一对alpha值发生变化
        0 - 没有任意一对alpha值发生变化或变化太小
    """
```

```
#步骤1: 计算误差Ei
Ei = calcEk(oS, i)
#优化alpha,设定一定的容错率。
if ((oS.labelMat[i] * Ei < -oS.tol) and (oS.alphas[i] < oS.C)) or ((oS.labelMat[i] * Ei > oS.tol)
    and (oS.alphas[i] > 0)):
    #使用内循环启发方式2选择alpha_j,并计算Ej
    j,Ej = selectJ(i, oS, Ei)
    #保存更新前的aplpha值, 使用深拷贝
    alphaIold = oS.alphas[i].copy(); alphaJold = oS.alphas[j].copy();
    #步骤2: 计算上下界L和H
    if (oS.labelMat[i] != oS.labelMat[j]):
        L = max(0, oS.alphas[j] - oS.alphas[i])
        H = min(oS.C, oS.C + oS.alphas[j] - oS.alphas[i])
    else:
        L = max(0, oS.alphas[j] + oS.alphas[i] - oS.C)
        H = min(oS.C, oS.alphas[j] + oS.alphas[i])
    if L == H:
        print("L==H")
        return 0
    #步骤3: 计算eta
    eta = 2.0 * oS.K[i,j] - oS.K[i,i] - oS.K[j,j]
    if eta >= 0:
        print("eta>=0")
        return 0
    #步骤4: 更新alpha_j
    oS.alphas[j] -= oS.labelMat[j] * (Ei - Ej)/eta
    #步骤5: 修剪alpha_j
    oS.alphas[j] = clipAlpha(oS.alphas[j],H,L)
    #更新Ej至误差缓存
    updateEk(oS, j)
    if (abs(oS.alphas[j] - alphaJold) < 0.00001):
        print("alpha_j变化太小")
        return 0
    #步骤6: 更新alpha_i
    oS.alphas[i] += oS.labelMat[j]*oS.labelMat[i]*(alphaJold - oS.alphas[j])
    #更新Ei至误差缓存
    updateEk(oS, i)
    #步骤7: 更新b_1和b_2
    b1 = oS.b - Ei- oS.labelMat[i]*(oS.alphas[i]-alphaIold)*oS.K[i,i] - oS.labelMat[j]*(oS.alphas[j
        ]-alphaJold)*oS.K[i,j]
    b2 = oS.b - Ej- oS.labelMat[i]*(oS.alphas[i]-alphaIold)*oS.K[i,j]- oS.labelMat[j]*(oS.alphas[j
        ]-alphaJold)*oS.K[j,j]
    #步骤8: 根据b_1和b_2更新b
    if (0 < oS.alphas[i]) and (oS.C > oS.alphas[i]): oS.b = b1
    elif (0 < oS.alphas[j]) and (oS.C > oS.alphas[j]): oS.b = b2
    else: oS.b = (b1 + b2)/2.0
    return 1
else:
    return 0
```

```
def calcEk(oS, k):
    """
    计算误差
    Parameters:
        oS - 数据结构
        k - 标号为k的数据
    Returns:
        Ek - 标号为k的数据误差
    """
    fXk = float(np.multiply(oS.alphas,oS.labelMat).T*oS.K[:,k] + oS.b)
    Ek = fXk - float(oS.labelMat[k])
    return Ek
```

10.5.2 在测试中使用核函数

接下来我们将构建一个对图10.4中的数据点进行有效分类的分类器，该分类器使用了径向基核函数。前面提到的径向基函数有一个用户定义的输入 。首先，我们需要确定它的大小，然后利用该核函数构建出一个分类器。

从学在浙大上下载数据。

```
# -*-coding:utf-8 -*-
import matplotlib.pyplot as plt
import numpy as np
import random

"""
Author:
    Jack Cui
Blog:
    http://blog.csdn.net/c406495762
Zhihu:
    https://www.zhihu.com/people/Jack--Cui/
Modify:
    2017-10-03
"""

class optStruct:
    """
    数据结构，维护所有需要操作的值
    Parameters:
        dataMatIn - 数据矩阵
        classLabels - 数据标签
        C - 松弛变量
        toler - 容错率
        kTup - 包含核函数信息的元组,第一个参数存放核函数类别，第二个参数存放必要的核函数需要用到的参数
    """
```

```
    def __init__(self, dataMatIn, classLabels, C, toler, kTup):
        self.X = dataMatIn                              #数据矩阵
        self.labelMat = classLabels                     #数据标签
        self.C = C                                      #松弛变量
        self.tol = toler                               #容错率
        self.m = np.shape(dataMatIn)[0]                #数据矩阵行数
        self.alphas = np.mat(np.zeros((self.m,1)))     #根据矩阵行数初始化alpha参数为0
        self.b = 0                                      #初始化b参数为0
        self.eCache = np.mat(np.zeros((self.m,2)))     #根据矩阵行数初始化虎误差缓存，第一列为是否有效的标
            志位，第二列为实际的误差E的值。
        self.K = np.mat(np.zeros((self.m,self.m)))    #初始化核K
        for i in range(self.m):                         #计算所有数据的核K
            self.K[:,i] = kernelTrans(self.X, self.X[i,:], kTup)

def kernelTrans(X, A, kTup):
    """
    通过核函数将数据转换更高维的空间
    Parameters:
        X - 数据矩阵
        A - 单个数据的向量
        kTup - 包含核函数信息的元组
    Returns:
        K - 计算的核K
    """
    m,n = np.shape(X)
    K = np.mat(np.zeros((m,1)))
    if kTup[0] == 'lin': K = X * A.T                    #线性核函数,只进行内积。
    elif kTup[0] == 'rbf':                              #高斯核函数,根据高斯核函数公式进行计算
        for j in range(m):
            deltaRow = X[j,:] - A
            K[j] = deltaRow*deltaRow.T
        K = np.exp(K/(-1*kTup[1]**2))                   #计算高斯核K
    else: raise NameError('核函数无法识别')
    return K                                           #返回计算的核K

def loadDataSet(fileName):
    """
    读取数据
    Parameters:
        fileName - 文件名
    Returns:
        dataMat - 数据矩阵
        labelMat - 数据标签
    """
    dataMat = []; labelMat = []
    fr = open(fileName)
    for line in fr.readlines():                                    #逐行读取，滤除空格等
        lineArr = line.strip().split('\t')
        dataMat.append([float(lineArr[0]), float(lineArr[1])]) #添加数据
```

```
        labelMat.append(float(lineArr[2]))                    #添加标签
    return dataMat,labelMat

def calcEk(oS, k):
    """
    计算误差
    Parameters:
        oS - 数据结构
        k - 标号为k的数据
    Returns:
        Ek - 标号为k的数据误差
    """
    fXk = float(np.multiply(oS.alphas,oS.labelMat).T*oS.K[:,k] + oS.b)
    Ek = fXk - float(oS.labelMat[k])
    return Ek

def selectJrand(i, m):
    """
    函数说明:随机选择alpha_j的索引值

    Parameters:
        i - alpha_i的索引值
        m - alpha参数个数
    Returns:
        j - alpha_j的索引值
    """
    j = i                                #选择一个不等于i的j
    while (j == i):
        j = int(random.uniform(0, m))
    return j

def selectJ(i, oS, Ei):
    """
    内循环启发方式2
    Parameters:
        i - 标号为i的数据的索引值
        oS - 数据结构
        Ei - 标号为i的数据误差
    Returns:
        j, maxK - 标号为j或maxK的数据的索引值
        Ej - 标号为j的数据误差
    """
    maxK = -1; maxDeltaE = 0; Ej = 0                    #初始化
    oS.eCache[i] = [1,Ei]                                 #根据Ei更新误差缓存
    validEcacheList = np.nonzero(oS.eCache[:,0].A)[0]   #返回误差不为0的数据的索引值
    if (len(validEcacheList)) > 1:                       #有不为0的误差
        for k in validEcacheList:                       #遍历,找到最大的Ek
            if k == i: continue                         #不计算i,浪费时间
            Ek = calcEk(oS, k)                           #计算Ek
```

```
            deltaE = abs(Ei - Ek)                          #计算|Ei-Ek|
            if (deltaE > maxDeltaE):                       #找到maxDeltaE
                maxK = k; maxDeltaE = deltaE; Ej = Ek
        return maxK, Ej                                    #返回maxK,Ej
    else:                                                  #没有不为0的误差
        j = selectJrand(i, oS.m)                           #随机选择alpha_j的索引值
        Ej = calcEk(oS, j)                                 #计算Ej
    return j, Ej                                           #j,Ej

def updateEk(oS, k):
    """
    计算Ek,并更新误差缓存
    Parameters:
        oS - 数据结构
        k - 标号为k的数据的索引值
    Returns:
        无
    """
    Ek = calcEk(oS, k)                                     #计算Ek
    oS.eCache[k] = [1,Ek]                                  #更新误差缓存

def clipAlpha(aj,H,L):
    """
    修剪alpha_j
    Parameters:
        aj - alpha_j的值
        H - alpha上限
        L - alpha下限
    Returns:
        aj - 修剪后的alpah_j的值
    """
    if aj > H:
        aj = H
    if L > aj:
        aj = L
    return aj

def innerL(i, oS):
    """
    优化的SMO算法
    Parameters:
        i - 标号为i的数据的索引值
        oS - 数据结构
    Returns:
        1 - 有任意一对alpha值发生变化
        0 - 没有任意一对alpha值发生变化或变化太小
    """
    #步骤1: 计算误差Ei
    Ei = calcEk(oS, i)
```

```
    #优化alpha,设定一定的容错率。
    if ((oS.labelMat[i] * Ei < -oS.tol) and (oS.alphas[i] < oS.C)) or ((oS.labelMat[i] * Ei > oS.tol)
         and (oS.alphas[i] > 0)):
        #使用内循环启发方式2选择alpha_j,并计算Ej
        j,Ej = selectJ(i, oS, Ei)
        #保存更新前的aplpha值，使用深拷贝
        alphaIold = oS.alphas[i].copy(); alphaJold = oS.alphas[j].copy();
        #步骤2：计算上下界L和H
        if (oS.labelMat[i] != oS.labelMat[j]):
            L = max(0, oS.alphas[j] - oS.alphas[i])
            H = min(oS.C, oS.C + oS.alphas[j] - oS.alphas[i])
        else:
            L = max(0, oS.alphas[j] + oS.alphas[i] - oS.C)
            H = min(oS.C, oS.alphas[j] + oS.alphas[i])
        if L == H:
            print("L==H")
            return 0
        #步骤3：计算eta
        eta = 2.0 * oS.K[i,j] - oS.K[i,i] - oS.K[j,j]
        if eta >= 0:
            print("eta>=0")
            return 0
        #步骤4：更新alpha_j
        oS.alphas[j] -= oS.labelMat[j] * (Ei - Ej)/eta
        #步骤5：修剪alpha_j
        oS.alphas[j] = clipAlpha(oS.alphas[j],H,L)
        #更新Ej至误差缓存
        updateEk(oS, j)
        if (abs(oS.alphas[j] - alphaJold) < 0.00001):
            print("alpha_j变化太小")
            return 0
        #步骤6：更新alpha_i
        oS.alphas[i] += oS.labelMat[j]*oS.labelMat[i]*(alphaJold - oS.alphas[j])
        #更新Ei至误差缓存
        updateEk(oS, i)
        #步骤7：更新b_1和b_2
        b1 = oS.b - Ei- oS.labelMat[i]*(oS.alphas[i]-alphaIold)*oS.K[i,i] - oS.labelMat[j]*(oS.alphas[j
            ]-alphaJold)*oS.K[i,j]
        b2 = oS.b - Ej- oS.labelMat[i]*(oS.alphas[i]-alphaIold)*oS.K[i,j]- oS.labelMat[j]*(oS.alphas[j
            ]-alphaJold)*oS.K[j,j]
        #步骤8：根据b_1和b_2更新b
        if (0 < oS.alphas[i]) and (oS.C > oS.alphas[i]): oS.b = b1
        elif (0 < oS.alphas[j]) and (oS.C > oS.alphas[j]): oS.b = b2
        else: oS.b = (b1 + b2)/2.0
        return 1
    else:
        return 0

def smoP(dataMatIn, classLabels, C, toler, maxIter, kTup = ('lin',0)):
```

```
    """
    完整的线性SMO算法
    Parameters:
        dataMatIn - 数据矩阵
        classLabels - 数据标签
        C - 松弛变量
        toler - 容错率
        maxIter - 最大迭代次数
        kTup - 包含核函数信息的元组
    Returns:
        oS.b - SMO算法计算的b
        oS.alphas - SMO算法计算的alphas
    """
    oS = optStruct(np.mat(dataMatIn), np.mat(classLabels).transpose(), C, toler, kTup)       #初始化数据
        结构
    iter = 0                                                                                 #初始化当前
        迭代次数
    entireSet = True; alphaPairsChanged = 0
    while (iter < maxIter) and ((alphaPairsChanged > 0) or (entireSet)):                     #遍历整个数
        据集都alpha也没有更新或者超过最大迭代次数,则退出循环
        alphaPairsChanged = 0
        if entireSet:                                                                        #遍历整个数
            据集
            for i in range(oS.m):
                alphaPairsChanged += innerL(i,oS)                                            #使用优化的
                    SMO算法
                print("全样本遍历:第%d次迭代 样本:%d, alpha优化次数:%d" % (iter,i,alphaPairsChanged))
            iter += 1
        else:                                                                                #遍历非边界
            值
            nonBoundIs = np.nonzero((oS.alphas.A > 0) * (oS.alphas.A < C))[0]                #遍历不在边
                界0和C的alpha
            for i in nonBoundIs:
                alphaPairsChanged += innerL(i,oS)
                print("非边界遍历:第%d次迭代 样本:%d, alpha优化次数:%d" % (iter,i,alphaPairsChanged))
            iter += 1
        if entireSet:                                                                        #遍历一次后
            改为非边界遍历
            entireSet = False
        elif (alphaPairsChanged == 0):                                                       #如果alpha
            没有更新,计算全样本遍历
            entireSet = True
        print("迭代次数: %d" % iter)
    return oS.b,oS.alphas                                                                    #返回SMO算
        法计算的b和alphas

def testRbf(k1 = 1.3):
    """
    测试函数
```

```
    Parameters:
        k1 - 使用高斯核函数的时候表示到达率
    Returns:
        无
    """
    dataArr,labelArr = loadDataSet('testSetRBF.txt')                    #加载训练集
    b,alphas = smoP(dataArr, labelArr, 200, 0.0001, 100, ('rbf', k1)) #根据训练集计算b和alphas
    datMat = np.mat(dataArr); labelMat = np.mat(labelArr).transpose()
    svInd = np.nonzero(alphas.A > 0)[0]                                  #获得支持向量
    sVs = datMat[svInd]
    labelSV = labelMat[svInd];
    print("支持向量个数:%d" % np.shape(sVs)[0])
    m,n = np.shape(datMat)
    errorCount = 0
    for i in range(m):
        kernelEval = kernelTrans(sVs,datMat[i,:],('rbf', k1))          #计算各个点的核
        predict = kernelEval.T * np.multiply(labelSV,alphas[svInd]) + b #根据支持向量的点，计算超平面，
            返回预测结果
        if np.sign(predict) != np.sign(labelArr[i]): errorCount += 1 #返回数组中各元素的正负符号，用1和
            -1表示，并统计错误个数
    print("训练集错误率: %.2f%%" % ((float(errorCount)/m)*100))       #打印错误率
    dataArr,labelArr = loadDataSet('testSetRBF2.txt')                   #加载测试集
    errorCount = 0
    datMat = np.mat(dataArr); labelMat = np.mat(labelArr).transpose()
    m,n = np.shape(datMat)
    for i in range(m):
        kernelEval = kernelTrans(sVs,datMat[i,:],('rbf', k1))           #计算各个点的核
        predict=kernelEval.T * np.multiply(labelSV,alphas[svInd]) + b  #根据支持向量的点，计算超平面，返
            回预测结果
        if np.sign(predict) != np.sign(labelArr[i]): errorCount += 1 #返回数组中各元素的正负符号，用1和
            -1表示，并统计错误个数
    print("测试集错误率: %.2f%%" % ((float(errorCount)/m)*100))       #打印错误率

def showDataSet(dataMat, labelMat):
    """
    数据可视化
    Parameters:
        dataMat - 数据矩阵
        labelMat - 数据标签
    Returns:
        无
    """
    data_plus = []                                  #正样本
    data_minus = []                                 #负样本
    for i in range(len(dataMat)):
        if labelMat[i] > 0:
            data_plus.append(dataMat[i])
        else:
            data_minus.append(dataMat[i])
```

```
    data_plus_np = np.array(data_plus)          #转换为numpy矩阵
    data_minus_np = np.array(data_minus)        #转换为numpy矩阵
    plt.scatter(np.transpose(data_plus_np)[0], np.transpose(data_plus_np)[1]) #正样本散点图
    plt.scatter(np.transpose(data_minus_np)[0], np.transpose(data_minus_np)[1]) #负样本散点图
    plt.show()

if __name__ == '__main__':
    testRbf()
```

第 10 章 练习

1. 基于 SVM 的手写数字识别

```
from numpy import *
from time import sleep

def loadDataSet(fileName):
    dataMat = []; labelMat = []
    fr = open(fileName)
    for line in fr.readlines():
        lineArr = line.strip().split('\t')
        dataMat.append([float(lineArr[0]), float(lineArr[1])])
        labelMat.append(float(lineArr[2]))
    return dataMat,labelMat

def selectJrand(i,m):
    j=i #we want to select any J not equal to i
    while (j==i):
        j = int(random.uniform(0,m))
    return j

def clipAlpha(aj,H,L):
    if aj > H:
        aj = H
    if L > aj:
        aj = L
    return aj

def smoSimple(dataMatIn, classLabels, C, toler, maxIter):
    dataMatrix = mat(dataMatIn); labelMat = mat(classLabels).transpose()
    b = 0; m,n = shape(dataMatrix)
    alphas = mat(zeros((m,1)))
    iter = 0
    while (iter < maxIter):
        alphaPairsChanged = 0
        for i in range(m):
            fXi = float(multiply(alphas,labelMat).T*(dataMatrix*dataMatrix[i,:].T)) + b
            Ei = fXi - float(labelMat[i])#if checks if an example violates KKT conditions
            if ((labelMat[i]*Ei < -toler) and (alphas[i] < C)) or ((labelMat[i]*Ei > toler) and (
                alphas[i] > 0)):
```

```
            j = selectJrand(i,m)
            fXj = float(multiply(alphas,labelMat).T*(dataMatrix*dataMatrix[j,:].T)) + b
            Ej = fXj - float(labelMat[j])
            alphaIold = alphas[i].copy(); alphaJold = alphas[j].copy();
            if (labelMat[i] != labelMat[j]):
                L = max(0, alphas[j] - alphas[i])
                H = min(C, C + alphas[j] - alphas[i])
            else:
                L = max(0, alphas[j] + alphas[i] - C)
                H = min(C, alphas[j] + alphas[i])
            if L==H: print ("L==H"); continue
            eta = 2.0 * dataMatrix[i,:]*dataMatrix[j,:].T - dataMatrix[i,:]*dataMatrix[i,:].T
                - dataMatrix[j,:]*dataMatrix[j,:].T
            if eta >= 0: print ("eta>=0"); continue
            alphas[j] -= labelMat[j]*(Ei - Ej)/eta
            alphas[j] = clipAlpha(alphas[j],H,L)
            if (abs(alphas[j] - alphaJold) < 0.00001): print ("j not moving enough"); continue
            alphas[i] += labelMat[j]*labelMat[i]*(alphaJold - alphas[j])#update i by the same
                amount as j
                                                    #the update is in the oppostie
                                                        direction
            b1 = b - Ei- labelMat[i]*(alphas[i]-alphaIold)*dataMatrix[i,:]*dataMatrix[i,:].T -
                 labelMat[j]*(alphas[j]-alphaJold)*dataMatrix[i,:]*dataMatrix[j,:].T
            b2 = b - Ej- labelMat[i]*(alphas[i]-alphaIold)*dataMatrix[i,:]*dataMatrix[j,:].T -
                 labelMat[j]*(alphas[j]-alphaJold)*dataMatrix[j,:]*dataMatrix[j,:].T
            if (0 < alphas[i]) and (C > alphas[i]): b = b1
            elif (0 < alphas[j]) and (C > alphas[j]): b = b2
            else: b = (b1 + b2)/2.0
            alphaPairsChanged += 1
            print ("iter: %d i:%d, pairs changed %d" % (iter,i,alphaPairsChanged))
        if (alphaPairsChanged == 0): iter += 1
        else: iter = 0
        print ("iteration number: %d" % iter)
    return b,alphas

def kernelTrans(X, A, kTup): #calc the kernel or transform data to a higher dimensional space
    m,n = shape(X)
    K = mat(zeros((m,1)))
    if kTup[0]=='lin': K = X * A.T #linear kernel
    elif kTup[0]=='rbf':
        for j in range(m):
            deltaRow = X[j,:] - A
            K[j] = deltaRow*deltaRow.T
        K = exp(K/(-1*kTup[1]**2)) #divide in NumPy is element-wise not matrix like Matlab
    else: raise NameError('Houston We Have a Problem -- \
    That Kernel is not recognized')
    return K

class optStruct:
```

```
    def __init__(self,dataMatIn, classLabels, C, toler, kTup): # Initialize the structure with
        the parameters
        self.X = dataMatIn
        self.labelMat = classLabels
        self.C = C
        self.tol = toler
        self.m = shape(dataMatIn)[0]
        self.alphas = mat(zeros((self.m,1)))
        self.b = 0
        self.eCache = mat(zeros((self.m,2))) #first column is valid flag
        self.K = mat(zeros((self.m,self.m)))
        for i in range(self.m):
            self.K[:,i] = kernelTrans(self.X, self.X[i,:], kTup)

def calcEk(oS, k):
    fXk = float(multiply(oS.alphas,oS.labelMat).T*oS.K[:,k] + oS.b)
    Ek = fXk - float(oS.labelMat[k])
    return Ek

def selectJ(i, oS, Ei):      #this is the second choice -heurstic, and calcs Ej
    maxK = -1; maxDeltaE = 0; Ej = 0
    oS.eCache[i] = [1,Ei] #set valid #choose the alpha that gives the maximum delta E
    validEcacheList = nonzero(oS.eCache[:,0].A)[0]
    if (len(validEcacheList)) > 1:
        for k in validEcacheList: #loop through valid Ecache values and find the one that
            maximizes delta E
            if k == i: continue #don't calc for i, waste of time
            Ek = calcEk(oS, k)
            deltaE = abs(Ei - Ek)
            if (deltaE > maxDeltaE):
                maxK = k; maxDeltaE = deltaE; Ej = Ek
        return maxK, Ej
    else:  #in this case (first time around) we don't have any valid eCache values
        j = selectJrand(i, oS.m)
        Ej = calcEk(oS, j)
    return j, Ej

def updateEk(oS, k):#after any alpha has changed update the new value in the cache
    Ek = calcEk(oS, k)
    oS.eCache[k] = [1,Ek]

def innerL(i, oS):
    Ei = calcEk(oS, i)
    if ((oS.labelMat[i]*Ei < -oS.tol) and (oS.alphas[i] < oS.C)) or ((oS.labelMat[i]*Ei > oS.tol)
        and (oS.alphas[i] > 0)):
        j,Ej = selectJ(i, oS, Ei) #this has been changed from selectJrand
        alphaIold = oS.alphas[i].copy(); alphaJold = oS.alphas[j].copy();
        if (oS.labelMat[i] != oS.labelMat[j]):
            L = max(0, oS.alphas[j] - oS.alphas[i])
```

```
            H = min(oS.C, oS.C + oS.alphas[j] - oS.alphas[i])
        else:
            L = max(0, oS.alphas[j] + oS.alphas[i] - oS.C)
            H = min(oS.C, oS.alphas[j] + oS.alphas[i])
        if L==H: print ("L==H"); return 0
        eta = 2.0 * oS.K[i,j] - oS.K[i,i] - oS.K[j,j] #changed for kernel
        if eta >= 0: print ("eta>=0"); return 0
        oS.alphas[j] -= oS.labelMat[j]*(Ei - Ej)/eta
        oS.alphas[j] = clipAlpha(oS.alphas[j],H,L)
        updateEk(oS, j) #added this for the Ecache
        if (abs(oS.alphas[j] - alphaJold) < 0.00001): print ("j not moving enough"); return 0
        oS.alphas[i] += oS.labelMat[j]*oS.labelMat[i]*(alphaJold - oS.alphas[j])#update i by the
            same amount as j
        updateEk(oS, i) #added this for the Ecache              #the update is in the oppostie
            direction
        b1 = oS.b - Ei- oS.labelMat[i]*(oS.alphas[i]-alphaIold)*oS.K[i,i] - oS.labelMat[j]*(oS.
            alphas[j]-alphaJold)*oS.K[i,j]
        b2 = oS.b - Ej- oS.labelMat[i]*(oS.alphas[i]-alphaIold)*oS.K[i,j]- oS.labelMat[j]*(oS.
            alphas[j]-alphaJold)*oS.K[j,j]
        if (0 < oS.alphas[i]) and (oS.C > oS.alphas[i]): oS.b = b1
        elif (0 < oS.alphas[j]) and (oS.C > oS.alphas[j]): oS.b = b2
        else: oS.b = (b1 + b2)/2.0
        return 1
    else: return 0

def smoP(dataMatIn, classLabels, C, toler, maxIter,kTup=('lin', 0)): #full Platt SMO
    oS = optStruct(mat(dataMatIn),mat(classLabels).transpose(),C,toler, kTup)
    iter = 0
    entireSet = True; alphaPairsChanged = 0
    while (iter < maxIter) and ((alphaPairsChanged > 0) or (entireSet)):
        alphaPairsChanged = 0
        if entireSet: #go over all
            for i in range(oS.m):
                alphaPairsChanged += innerL(i,oS)
                print ("fullSet, iter: %d i:%d, pairs changed %d" % (iter,i,alphaPairsChanged))
            iter += 1
        else:#go over non-bound (railed) alphas
            nonBoundIs = nonzero((oS.alphas.A > 0) * (oS.alphas.A < C))[0]
            for i in nonBoundIs:
                alphaPairsChanged += innerL(i,oS)
                print ("non-bound, iter: %d i:%d, pairs changed %d" % (iter,i,alphaPairsChanged))
            iter += 1
        if entireSet: entireSet = False #toggle entire set loop
        elif (alphaPairsChanged == 0): entireSet = True
        print ("iteration number: %d" % iter)
    return oS.b,oS.alphas

def calcWs(alphas,dataArr,classLabels):
    X = mat(dataArr); labelMat = mat(classLabels).transpose()
```

```
    m,n = shape(X)
    w = zeros((n,1))
    for i in range(m):
        w += multiply(alphas[i]*labelMat[i],X[i,:].T)
    return w

def testRbf(k1=1.3):
    dataArr,labelArr = loadDataSet('testSetRBF.txt')
    b,alphas = smoP(dataArr, labelArr, 200, 0.0001, 10000, ('rbf', k1)) #C=200 important
    datMat=mat(dataArr); labelMat = mat(labelArr).transpose()
    svInd=nonzero(alphas.A>0)[0]
    sVs=datMat[svInd] #get matrix of only support vectors
    labelSV = labelMat[svInd];
    print ("there are %d Support Vectors" % shape(sVs)[0])
    m,n = shape(datMat)
    errorCount = 0
    for i in range(m):
        kernelEval = kernelTrans(sVs,datMat[i,:],('rbf', k1))
        predict=kernelEval.T * multiply(labelSV,alphas[svInd]) + b
        if sign(predict)!=sign(labelArr[i]): errorCount += 1
    print ("the training error rate is: %f" % (float(errorCount)/m))
    dataArr,labelArr = loadDataSet('testSetRBF2.txt')
    errorCount = 0
    datMat=mat(dataArr); labelMat = mat(labelArr).transpose()
    m,n = shape(datMat)
    for i in range(m):
        kernelEval = kernelTrans(sVs,datMat[i,:],('rbf', k1))
        predict=kernelEval.T * multiply(labelSV,alphas[svInd]) + b
        if sign(predict)!=sign(labelArr[i]): errorCount += 1
    print ("the test error rate is: %f" % (float(errorCount)/m))

def img2vector(filename):
    returnVect = zeros((1,1024))
    fr = open(filename)
    for i in range(32):
        lineStr = fr.readline()
        for j in range(32):
            returnVect[0,32*i+j] = int(lineStr[j])
    return returnVect

def loadImages(dirName):
    from os import listdir
    hwLabels = []
    trainingFileList = listdir(dirName)       #load the training set
    m = len(trainingFileList)
    trainingMat = zeros((m,1024))
    for i in range(m):
        fileNameStr = trainingFileList[i]
        fileStr = fileNameStr.split('.')[0] #take off .txt
```

```
        classNumStr = int(fileStr.split('_')[0])
        if classNumStr == 9: hwLabels.append(-1)
        else: hwLabels.append(1)
        trainingMat[i,:] = img2vector('%s/%s' % (dirName, fileNameStr))
    return trainingMat, hwLabels

def testDigits(kTup=('rbf', 10)):
    dataArr,labelArr = loadImages('trainingDigits')
    b,alphas = smoP(dataArr, labelArr, 200, 0.0001, 10000, kTup)
    datMat=mat(dataArr); labelMat = mat(labelArr).transpose()
    svInd=nonzero(alphas.A>0)[0]
    sVs=datMat[svInd]
    labelSV = labelMat[svInd];
    print ("there are %d Support Vectors" % shape(sVs)[0])
    m,n = shape(datMat)
    errorCount = 0
    for i in range(m):
        kernelEval = kernelTrans(sVs,datMat[i,:],kTup)
        predict=kernelEval.T * multiply(labelSV,alphas[svInd]) + b
        if sign(predict)!=sign(labelArr[i]): errorCount += 1
    print ("the training error rate is: %f" % (float(errorCount)/m))
    dataArr,labelArr = loadImages('testDigits')
    errorCount = 0
    datMat=mat(dataArr); labelMat = mat(labelArr).transpose()
    m,n = shape(datMat)
    for i in range(m):
        kernelEval = kernelTrans(sVs,datMat[i,:],kTup)
        predict=kernelEval.T * multiply(labelSV,alphas[svInd]) + b
        if sign(predict)!=sign(labelArr[i]): errorCount += 1
    print ("the test error rate is: %f" % (float(errorCount)/m))

'''########************************************
Non-Kernel VErsions below
'''########************************************

class optStructK:
    def __init__(self,dataMatIn, classLabels, C, toler): # Initialize the structure with the
         parameters
        self.X = dataMatIn
        self.labelMat = classLabels
        self.C = C
        self.tol = toler
        self.m = shape(dataMatIn)[0]
        self.alphas = mat(zeros((self.m,1)))
        self.b = 0
        self.eCache = mat(zeros((self.m,2))) #first column is valid flag

def calcEkK(oS, k):
    fXk = float(multiply(oS.alphas,oS.labelMat).T*(oS.X*oS.X[k,:].T)) + oS.b
```

```
    Ek = fXk - float(oS.labelMat[k])
    return Ek

def selectJK(i, oS, Ei):      #this is the second choice -heurstic, and calcs Ej
    maxK = -1; maxDeltaE = 0; Ej = 0
    oS.eCache[i] = [1,Ei] #set valid #choose the alpha that gives the maximum delta E
    validEcacheList = nonzero(oS.eCache[:,0].A)[0]
    if (len(validEcacheList)) > 1:
        for k in validEcacheList: #loop through valid Ecache values and find the one that
            maximizes delta E
            if k == i: continue #don't calc for i, waste of time
            Ek = calcEk(oS, k)
            deltaE = abs(Ei - Ek)
            if (deltaE > maxDeltaE):
                maxK = k; maxDeltaE = deltaE; Ej = Ek
        return maxK, Ej
    else:  #in this case (first time around) we don't have any valid eCache values
        j = selectJrand(i, oS.m)
        Ej = calcEk(oS, j)
    return j, Ej

def updateEkK(oS, k):#after any alpha has changed update the new value in the cache
    Ek = calcEk(oS, k)
    oS.eCache[k] = [1,Ek]

def innerLK(i, oS):
    Ei = calcEk(oS, i)
    if ((oS.labelMat[i]*Ei < -oS.tol) and (oS.alphas[i] < oS.C)) or ((oS.labelMat[i]*Ei > oS.tol)
        and (oS.alphas[i] > 0)):
        j,Ej = selectJ(i, oS, Ei) #this has been changed from selectJrand
        alphaIold = oS.alphas[i].copy(); alphaJold = oS.alphas[j].copy();
        if (oS.labelMat[i] != oS.labelMat[j]):
            L = max(0, oS.alphas[j] - oS.alphas[i])
            H = min(oS.C, oS.C + oS.alphas[j] - oS.alphas[i])
        else:
            L = max(0, oS.alphas[j] + oS.alphas[i] - oS.C)
            H = min(oS.C, oS.alphas[j] + oS.alphas[i])
        if L==H: print ("L==H"); return 0
        eta = 2.0 * oS.X[i,:]*oS.X[j,:].T - oS.X[i,:]*oS.X[i,:].T - oS.X[j,:]*oS.X[j,:].T
        if eta >= 0: print ("eta>=0"); return 0
        oS.alphas[j] -= oS.labelMat[j]*(Ei - Ej)/eta
        oS.alphas[j] = clipAlpha(oS.alphas[j],H,L)
        updateEk(oS, j) #added this for the Ecache
        if (abs(oS.alphas[j] - alphaJold) < 0.00001): print ("j not moving enough"); return 0
        oS.alphas[i] += oS.labelMat[j]*oS.labelMat[i]*(alphaJold - oS.alphas[j])#update i by the
            same amount as j
        updateEk(oS, i) #added this for the Ecache              #the update is in the oppostie
            direction
        b1 = oS.b - Ei- oS.labelMat[i]*(oS.alphas[i]-alphaIold)*oS.X[i,:]*oS.X[i,:].T - oS.
```

```
            labelMat[j]*(oS.alphas[j]-alphaJold)*oS.X[i,:]*oS.X[j,:].T
        b2 = oS.b - Ej- oS.labelMat[i]*(oS.alphas[i]-alphaIold)*oS.X[i,:]*oS.X[j,:].T - oS.
            labelMat[j]*(oS.alphas[j]-alphaJold)*oS.X[j,:]*oS.X[j,:].T
        if (0 < oS.alphas[i]) and (oS.C > oS.alphas[i]): oS.b = b1
        elif (0 < oS.alphas[j]) and (oS.C > oS.alphas[j]): oS.b = b2
        else: oS.b = (b1 + b2)/2.0
        return 1
    else: return 0

def smoPK(dataMatIn, classLabels, C, toler, maxIter): #full Platt SMO
    oS = optStruct(mat(dataMatIn),mat(classLabels).transpose(),C,toler)
    iter = 0
    entireSet = True; alphaPairsChanged = 0
    while (iter < maxIter) and ((alphaPairsChanged > 0) or (entireSet)):
        alphaPairsChanged = 0
        if entireSet: #go over all
            for i in range(oS.m):
                alphaPairsChanged += innerL(i,oS)
                print ("fullSet, iter: %d i:%d, pairs changed %d" % (iter,i,alphaPairsChanged))
            iter += 1
        else:#go over non-bound (railed) alphas
            nonBoundIs = nonzero((oS.alphas.A > 0) * (oS.alphas.A < C))[0]
            for i in nonBoundIs:
                alphaPairsChanged += innerL(i,oS)
                print ("non-bound, iter: %d i:%d, pairs changed %d" % (iter,i,alphaPairsChanged))
            iter += 1
        if entireSet: entireSet = False #toggle entire set loop
        elif (alphaPairsChanged == 0): entireSet = True
        print ("iteration number: %d" % iter)
    return oS.b,oS.alphas
```

```
[Parallel(n_jobs=2)]: Done 46 tasks   | elapsed: 27.3min
[Parallel(n_jobs=2)]: Done 90 out of 90 | elapsed: 53.4min finished
Best score: 0.963
Best parameters set:
  clf__C: 10
  clf__gamma: 0.01
             precision   recall f1-score  support

        0.0       0.98     0.99     0.98     1697
        1.0       0.99     0.98     0.98     2045
        2.0       0.95     0.98     0.96     1743
        3.0       0.96     0.96     0.96     1835
        4.0       0.96     0.97     0.97     1693
        5.0       0.97     0.96     0.97     1564
        6.0       0.98     0.99     0.98     1683
        7.0       0.98     0.97     0.97     1807
        8.0       0.97     0.96     0.96     1692
        9.0       0.97     0.95     0.96     1741
```

```
avg / total      0.97      0.97      0.97     17500
```

10.5.3 sklearn 分类字符

MNIST 数据集包含 70000 张手写数字图片的集合。这些图片是灰色图像，大小为 28×28。从学在浙大上下载数据：查看其中的照片：

```
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_mldata
import matplotlib.cm as cm

mnist = fetch_mldata('MNIST original', data_home='data/mnist')

counter = 1
for i in range(1, 4):
    for j in range(1, 6):
        plt.subplot(3, 5, counter)
        plt.imshow(mnist.data[(i - 1) * 8000 + j].reshape((28, 28)), cmap=cm.Greys_r)
        plt.axis('off')
        counter += 1
plt.show()
```

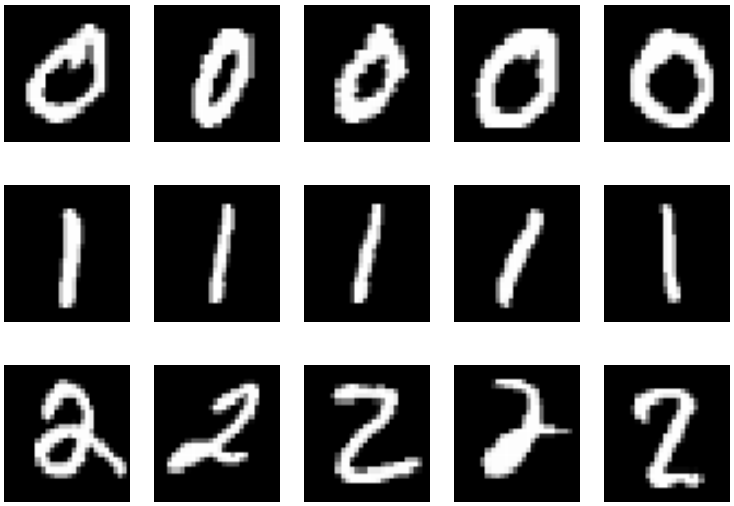

图 10.5: MNIST 数据集示例

MNIST 手写数字数据集包含 60000 张训练图像，10000 张测试图像。

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import scale
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.grid_search import GridSearchCV
from sklearn.metrics import classification_report

if __name__ == '__main__':
```

```
X, y = mnist.data, mnist.target
X = X/255.0*2 - 1
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=11)

pipeline = Pipeline([
    ('clf', SVC(kernel='rbf', gamma=0.01, C=100))
])

parameters = {
    'clf__gamma': (0.01, 0.03, 0.1, 0.3, 1),
    'clf__C': (0.1, 0.3, 1, 3, 10, 30),
}

grid_search = GridSearchCV(pipeline, parameters, n_jobs=2, verbose=1, scoring='accuracy')
grid_search.fit(X_train[:10000], y_train[:10000])
print('Best score: %0.3f' % grid_search.best_score_)
print('Best parameters set:')
best_parameters = grid_search.best_estimator_.get_params()
for param_name in sorted(parameters.keys()):
    print('\t%s: %r' % (param_name, best_parameters[param_name]))

predictions = grid_search.predict(X_test)
print(classification_report(y_test, predictions))
```

第 10 章 练习

1. 自然图片分类

   Chars74K 数据集 包括 74000 张图片，包括 0~9 以及英文大小写字母的字符。

```
import os
import numpy as np
from sklearn.pipeline import Pipeline
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.grid_search import GridSearchCV
from sklearn.metrics import classification_report
from PIL import Image


X = []
y = []
for path, subdirs, files in os.walk('data/English/Img/GoodImg/Bmp/'):
    for filename in files:
        f = os.path.join(path, filename)
        target = filename[3:filename.index('-')]
        img = Image.open(f).convert('L').resize((30, 30), resample=Image.LANCZOS)
        X.append(np.array(img).reshape(900,))
        y.append(target)
X = np.array(X)
```

剩余代码请独立完成。

图 10.6: Chars74K 数据集。