6.1 SMO 高效优化算法

1996年， John Platt发布了一个称为SMO的强大算法，用于训练SVM。 SMO表示序列最小优化（Sequential Minimal Optimization）。

- SMO目标：SMO算法的目标是求出一系列alpha和b，一旦求出了这些alpha，就很容易计算出权重向量w并得到分隔超平面
- SMO思想：Platt的SMO算法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原理：每次循环中选择两个alpha进行优化处理。一旦找到一对合适的alpha，那么就增大其中一个同时减小另一个.
    - 这里指的合适必须要符合一定的条件
        1. 这两个 alpha 必须要在间隔边界之外
        2. 这两个 alpha 还没有进行过区间化处理或者不在边界上。
    - 之所以要同时改变2个 alpha；原因是我们有一个约束条件： $(\sum_{i=1}^{m} a_i \cdot label_i = 0)$；如果只是修改一个 alpha，很可能导致约束条件失效。

SMO 算法的伪代码如下所示:

```
创建一个 alpha 向量并将其初始化为0向量
当迭代次数小于最大迭代次数时(外循环)
    对数据集中的每个数据向量(内循环)：
        如果该数据向量可以被优化
            随机选择另外一个数据向量
            同时优化这两个向量
            如果两个向量都不能被优化，退出内循环
    如果所有向量都没被优化，增加迭代数目，继续下一次循环
```

6.2 SVM开发流程

- 收集数据：可以使用任意方法。
- 准备数据：需要数值型数据。
- 分析数据：有助于可视化分隔超平面。
- 训练算法：SVM的大部分时间都源自训练，该过程主要实现两个参数的调优。
- 测试算法：十分简单的计算过程就可以实现。
- 使用算法：几乎所有分类问题都可以使用SVM，值得一提的是，SVM本身是一个二类分类器，对多类问题应SVM需要对代码做一些修改。

6.3 SVM算法的特点

- 优点：泛化错误率低，计算开销不大，结果易解释。

- 缺点：对参数调节和核函数的选择敏感，原始分类器不加修改仅适用于处理二类问题。

  适用数据类型：数值型和标称型数据。

6.4 项目1——应用简化版 SMO 算法处理小规模数据集

6.4.1 项目概述

对小规模数据点进行分类

Platt SMO算法中的外循环确定要优化的最佳alpha对。而简化版却会跳过这一部分，首先在数据集上遍历每一个alpha，然后在剩下的alpha集合中随机选择另一个alpha，从而构建alpha对。这里有一点相当重要，就是我们要同时改变两个alpha。之所以这样做是因为我们有一个约束条件：

$$\sum \alpha_i \cdot \text{label}^{(i)} = 0$$

由于改变一个alpha可能会导致该约束条件失效，因此我们总是同时改变两个alpha。

6.4.2 具体实现过程

一 SMO算法中的辅助函数 (loadDataSet、selectJrand和clipAlpha函数)

先看下数据的格式:

```
3.542485    1.977398    -1
3.018896    2.556416    -1
7.551510    -1.580030   1
2.114999    -0.004466   -1
8.127113    1.274372    1
7.108772    -0.986906   1
8.610639    2.046708    1
2.326297    0.265213    -1
3.634009    1.730537    -1
0.341367    -0.894998   -1
3.125951    0.293251    -1
2.123252    -0.783563   -1
0.887835    -2.797792   -1
7.139979    -2.329896   1
1.696414    -1.212496   -1
8.117032    0.623493    1
8.497162    -0.266649   1
4.658191    3.507396    -1
8.197181    1.545132    1
1.208047    0.213100    -1
1.928486    -0.321870   -1
2.175808    -0.014527   -1
7.886608    0.461755    1
3.223038    -0.552392   -1
3.628502    2.190585    -1
7.407860    -0.121961   1
7.286357    0.251077    1
2.301095    -0.533988   -1
-0.232542   -0.547690   -1
3.457096    -0.082216   -1
3.023938    -0.057392   -1
8.015003    0.885325    1
8.991748    0.923154    1
7.916831    -1.781735   1
7.616862    -0.217958   1
2.450939    0.744967    -1
7.270337    -2.507834   1
```

接下来准备数据:

具体代码如下:

```python
import numpy as np
import random

"""
函数说明: 读取数据

Parameters:
    fileName - 文件名
Returns:
    dataMat - 数据矩阵(list类型)
    labelMat - 数据标签(list类型)
"""

def loadDataSet(fileName):
    #初始化返回值
    dataMat = []; labelMat = []
```

```
    with open(fileName) as f:
        for line in f.readlines():
            lineArr = line.strip().split('\t')
            dataMat.append([float(lineArr[0]), float(lineArr[1])])
            labelMat.append(float(lineArr[-1]))
    return dataMat, labelMat

"""
函数说明: 随机选择alpha

Parameters:
    i - 第一个alpha的下标
    m - 所有alpha的数目
Returns:
    j - 不等于i的j
"""
def selectJrand(i, m):
    #选择一个不等于i的j
    j = i
    while (j == i):
        j = int(random.uniform(0, m))
    return j

"""
函数说明: 调整大于H或小于L的alpha值

Parameters:
    aj - alpha值
    H - alpha上界
    L - alpha下界
Returns:
    aj - alpha值
"""
def clipAlpha(aj, H, L):
    if aj > H:
        aj = H
    if aj < L:
        aj = L
    return aj
```

代码运行的结果就不展示了, 和之前章节的差不多

二 简化版的SMO算法

这里我和机器学习实战的书上的稍微有些不同, 关于eta的计算.

机器学习实战中的eta其实是统计学习方法中的eta值的负值, 因此为了保持和统计学习方法笔记中的一致性, 我将eta调整为和前面的笔记一致.

紧接着就会造成两个变化:

1. 代码中的第136行: `if eta <= 0` , 原书中是 `if eta >= 0`
2. 代码中的第141行: `alphas[j] += labelMat[j]*(Ei - Ej)/eta` , 原书中是 `alphas[j] -= labelMat[j]*(Ei - Ej)/eta`

同时,还有一个注意点就是:

关于两个列向量相乘, $\boldsymbol{x}_i^T * \boldsymbol{x}_j = \boldsymbol{x}_j^T * \boldsymbol{x}_i$, 因此关于b1, b2的计算我也调整了下, 和笔记一致. 实质上两者都是等价的.

更为详细的代码理解和解释, 都在注释中.

具体代码:

```
import numpy as np
import random

"""
函数说明: 读取数据

Parameters:
    fileName - 文件名
Returns:
    dataMat - 数据矩阵(list类型)
    labelMat - 数据标签(list类型)
"""

def loadDataSet(fileName):
    #初始化返回值
    dataMat = []; labelMat = []
    with open(fileName) as f:
        for line in f.readlines():
            lineArr = line.strip().split('\t')
            dataMat.append([float(lineArr[0]), float(lineArr[1])])
            labelMat.append(float(lineArr[-1]))
    return dataMat, labelMat

"""
函数说明: 随机选择alpha

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

"""
函数说明: 修剪alpha

Parameters:
    aj - alpha值
    H - alpha上界
    L - alpha下界
Returns:
```

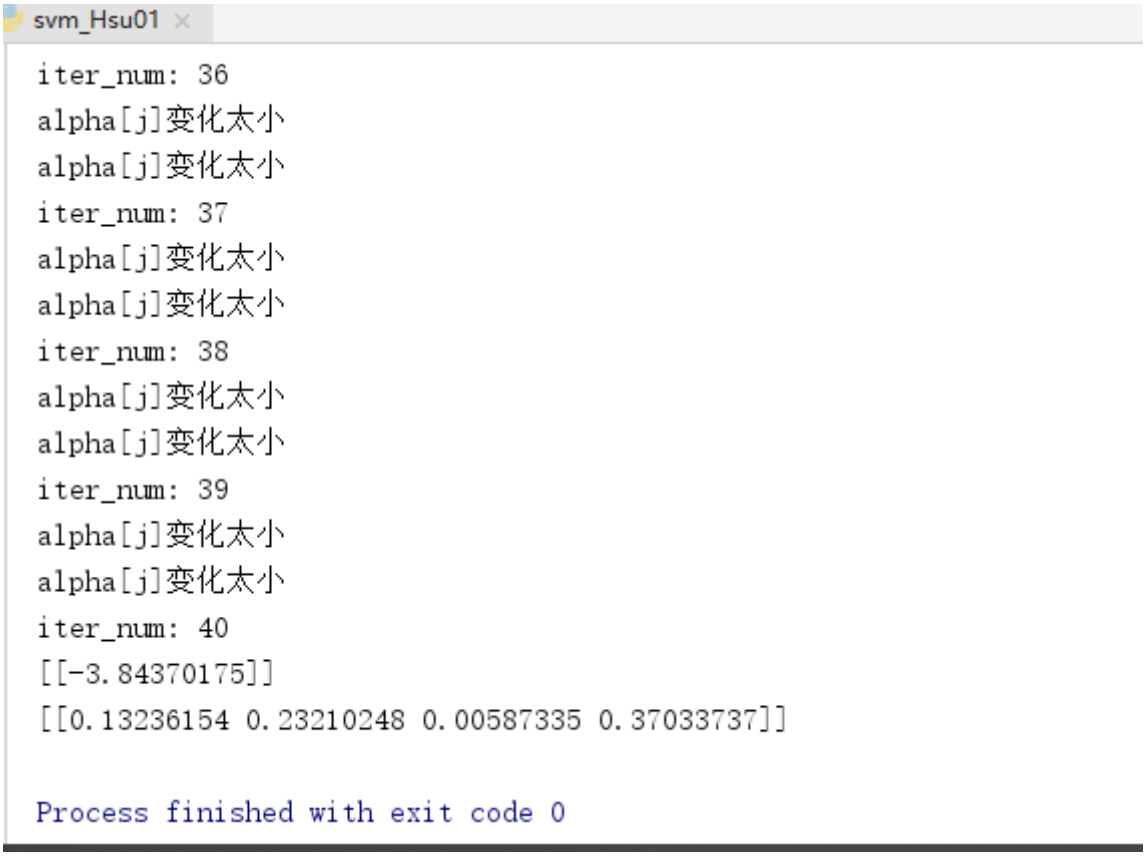
```
    aj - alpha值
"""
def clipAlpha(aj, H, L):
    if aj > H:
        aj = H
    if aj < L:
        aj = L
    return aj


"""
函数说明: 简化版的SMO算法

Parameters:
    dataMatIn - 数据矩阵
    classLabels - 数据标签
    C - 惩罚参数
    toler - 松弛变量
    maxIter - 最大迭代次数
Returns:
    b - 模型的常量值
    alphas - 拉格朗日乘子
"""
def smoSimple(dataMatIn, classLabels, C, toler, maxIter):
    #转换为numpy的matrix存储
    dataMatrix = np.mat(dataMatIn); labelMat = np.mat(classLabels).transpose()
    #初始化参数b,统计dataMatirx的维度
    b = 0; m, n = np.shape(dataMatrix)
    #初始化alpha参数为0
    alphas = np.mat(np.zeros((m,1)))
    #初始化迭代次数
    iter_num = 0
    # 限制循环迭代次数，也就是在数据集上遍历maxIter次，且不再发生任何alpha修改，则循环停止
    while (iter_num < maxIter):
        #记录alpha是否已经进行优化，每次循环时设为0，然后再对整个集合顺序遍历
        # 每次循环时先设为0，然后再对整个集合顺序遍历，该变量用于记录alpha是否已经进行优化
        alphaPairsChanged = 0
        for i in range(m):
            #计算误差Ei(根据公式可计算)
            #其实就是公式的实现，不过求和是以向量相乘的方式实现的
            fXi = float(np.multiply(alphas, labelMat).T * (dataMatrix *
dataMatrix[i,:].T)) + b
            #预测结果与真实结果比对，计算误差Ei
            Ei = fXi - float(labelMat[i])

            #如果误差很大就对该数据对应的alpha进行优化，正负间隔都会被测试，同时检查alpha值
            """
            把kkt条件写下:
             yi*f(i) >= 1 and alpha = 0 (outside the boundary)
            yi*f(i) == 1 and 0<alpha< C (on the boundary)
            yi*f(i) <= 1 and alpha = C (between the boundary)

            为什么小于 -toler时要同时满足alphas[i] < C以及后面的大于toler时,同时满足
alphas[i] > 0
            观察一下kkt条件可以看到,fi*f(i)越大，那么alpha的值越小，那么就可以知道原因为何
了.
            """
```

```
            if ((labelMat[i]*Ei < -toler) and (alphas[i] < C) or
((labelMat[i]*Ei) > toler) and (alphas[i] > 0)):
                # 随机选择不等于i的0-m的第二个alpha值
                j = selectJrand(i,m)
                #步骤1：计算误差Ej，和上面的步骤一样的
                fXj = float(np.multiply(alphas, labelMat).T *
(dataMatrix*dataMatrix[j,:].T)) + b
                Ej = fXj - float(labelMat[j])
                #保存更新前的alpha值，以备后用(深拷贝)
                alphaIold = alphas[i].copy(); alphaJold = alphas[j].copy()

                #步骤2：计算上界L和下界H
                # 这里是对SMO最优化问题的子问题的约束条件的分析
                # L和H用于将alphas[j]调整到0-C之间。如果L==H，就不做任何改变，直接执行
continue语句
                # 如果labelMat[i] != labelMat[j] 表示异侧，就相减，否则是同侧，就相加。
                #下面就是L和H的计算公式,参见之前的笔记
                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
                #计算eta(eta是模,为正的.其实这里的eta(机器学习实战),是实际eta(统计学习方
法)的负值)
                #同时为了简化计算,当eta=0时停止循环
                #同时要保证eta>0
                #注意这里，我把eta的符号变更了，后面要注意符号,其实这样可以更助于我们理解书
上的公式计算
                """
               同时,这点我之前没特别注意，西瓜书上的时x_{i}^{T}*x_{j} ，要注意x_{i}或者
x_{j}都是列向量，而我们的
               dataMatrix[i,:]是行向量，所以变换成
了:dataMatrix[i,:]*dataMatrix[j,:].T
               其实都是一种写作的方式,一般情况下，一行数据表示一个样本.
                """
                eta = -2 * 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
                #并使用辅助函数，对L和H进行调整
                alphas[j] = clipAlpha(alphas[j], H, L)
                # 检查alpha[j]是否只是轻微的改变，如果是的话，就退出for循环
                if (abs(alphas[j] - alphaJold) < 0.00001):
                    print("alpha[j]变化太小")
                    continue

                #步骤5：更新alpha[i]
                #根据前面的笔记易得
                #这里是简化版的，没有剪辑
```

```
                alphas[i] += labelMat[j]*labelMat[i]*(alphaJold-alphas[j])

                #步骤6: 更新b1和b2
                #注意b1和b2最后一个表达式与书中不一样, 实质是一样的, 都是1xm * mx1矩阵,写成下面这样,我觉得可以与公式统一
                #也就是dataMatrix[i,:]*dataMatrix[j,:].T变成了dataMatrix[j,:]*dataMatrix[i,:].T, 实质上没有变化
                b1 = b - Ei - labelMat[i]*(alphas[i]-alphaIold)*dataMatrix[i,:]*dataMatrix[i,:].T - labelMat[j]*(alphas[j]-alphaJold)*dataMatrix[j,:]*dataMatrix[i,:].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7:根据b1和b2更新b
                #参考前面的笔记
                #当然,在python中还可以直接写成: 0 < alphas[i] < C.
                if (0 < alphas[i]) and (alphas[i] < C):
                    b = b1
                elif (0 < alphas[j]) and (alphas[j] < C):
                    b = b2
                else:
                    b = (b1 + b2) / 2

                #到上面那里的话说明已经成功改变了一对alpha了
                #优化次数
                alphaPairsChanged += 1
                print("iter: %d i:%d, pairs changed %d" % (iter_num, i, alphaPairsChanged))

        #更新迭代次数
        if (alphaPairsChanged == 0):
            iter_num += 1
        else:
            iter_num = 0
        print("iter_num: %d" % iter_num)
    return b, alphas



if __name__ == '__main__':
    dataMatIn, classLabels = loadDataSet('testSet.txt')
    b, alphas = smoSimple(dataMatIn, classLabels, 0.6, 0.001, 40)
    print(b)
    print(alphas[alphas > 0])
```

还有一个, 因为返回的alphas中很多都是0, 所以选择打印出alphas中大于0的矩阵

结果如下:



一个简化版的SMO算法都看了这么长时间, 后面的完整版的SMO算法,以及非线性的支持向量机, 也就是添加了核函数的支持向量机, 不知道要看多久.

本来是打算全部弄完再发的, 这不马上中秋节了吗, 已经4,5个月没回家的我, 准备回去啦. 估计也弄不了什么东西, 所以, 就把这两天弄的先贴出来. 回来再继续吧.

中秋快乐.