

**模式识别大作业**

题 目 C++实现的SVM分类器

学 院 信息科学与工程

专 业 信息与通信工程

组 员 杨文友

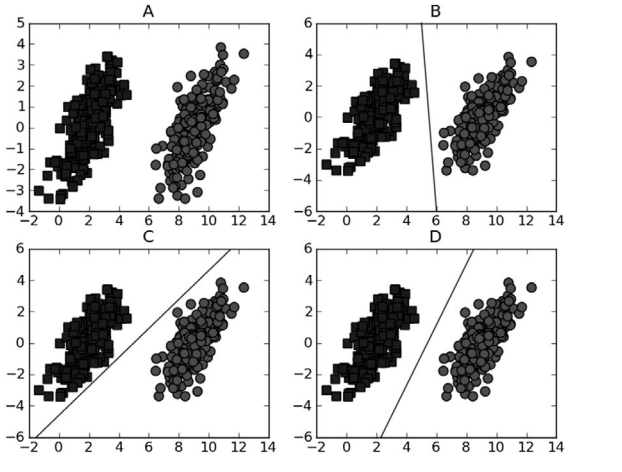
指导教师 赵海涛

**完成日期： 2019 年 11 月20日**

**模式识别作业报告——C++实现的SVM分类器**

在本学期的模式识别课程里学到了支持向量机（Support Vector Machine, SVM）,对其中的许多概念还是不太清楚，于是选择了C++实现SVM，并对其中涉及到的概念重新进行梳理，参考网上的相关资源实现了一个基于QT实现的SVM可视化GUI程序，源码位于<https://github.com/yangwyou/svm_Visualization>。

1. **SVM**

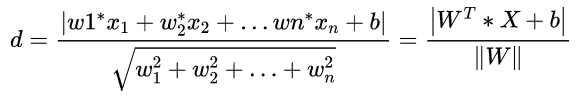
SVM（Support Vector Machines，支持向量机）是在所有知名的数据挖掘算法中最健壮、最准确的方法之一，它属于二分类算法，可以支持线性和非线性的分类。  
**1.支持向量与超平面**  
 在了解SVM算法之前，首先介绍线性分类器的有关概念。比如给定一系列的数据样本，每个样本都有对应的一个标签。为了使得描述更加直观，我们采用二维平面进行解释，高维空间原理也是一样。假设在一个二维线性可分的数据集中，图一A所示，我们要找到一个超平面把两组数据分开，这时，我们认为线性回归的直线或逻辑回归的直线也能够做这个分类，这条直线可以是图一B中的直线，也可以是图一C中的直线，或者图D中的直线，在高维空间里这条线就相当于我们要找的超平面。  


（图片来源：<https://www.jianshu.com/p/bcf069ff9dd3>）

2)有关的数学推导

超平面可以写成：

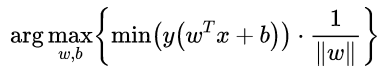
2.1）**点到平面的距离**

有了超平面的表达式之后之后，我们就可以计算样本点到平面的距离了。假设P(为样本点中的一个点，其中为第i个特征向量。那么该点到超平面的距离可以用如下公式进行计算：

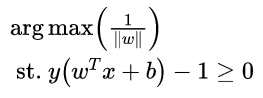
其中||W||为超平面的2范式距离，常数b类似于直线方程中的截距。

2.2）**最大间隔的优化模型**

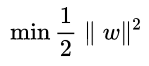
每一个超平面都对应着一个margin，我们的目标就是找出所有margin中最大的那个值对应的超平面。因此用数学语言描述就是确定w、b使得margin最大。这是一个优化问题其目标函数可以写成：

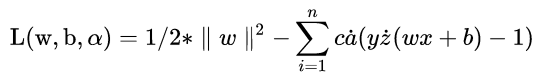


其中y表示数据点的标签，且其为-1或1。距离用计算y\**(w*x+b)，这是就能体会出-1和1的好处了。如果数据点在平面的正方向(即+1类)那么y\**(w*x+b)是一个正数，而当数据点在平面的负方向时(即-1类)，y*(w*x+b)依然是一个正数，这样就能够保证始终大于零了。注意到当w和b等比例放大时，d的结果是不会改变的。因此我们可以令所有支持向量的u为1，而其他点的u大1这是可以办通过调节w和b求到的。因此上面的问题可以简化为：

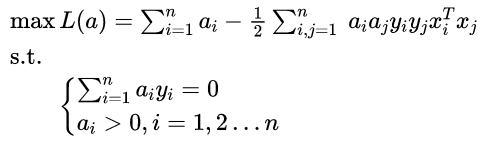


为了后面计算的方便，目标函数等价替换为：

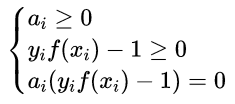


这是一个有约束条件的优化问题，通常我们可以用拉格朗日乘子法来求解。应用拉格朗日乘子法如下：

求偏导数化简，把原始问题转换为对偶问题：



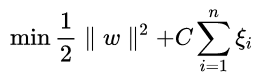
该对偶问题的KKT条件为：



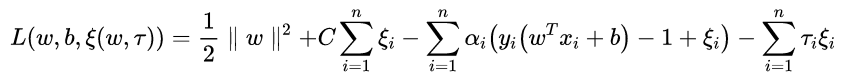
到此，问题并没有完美地解决。因为这里假设数据是可分的，但是实际中的数据几乎都不那么“干净”，或多或少都会存在一些噪点。为此引入了松弛变量来解决这种问题。这时新的约束条件变为:



所以新的目标函数变为：



新的拉格朗日函数变为：



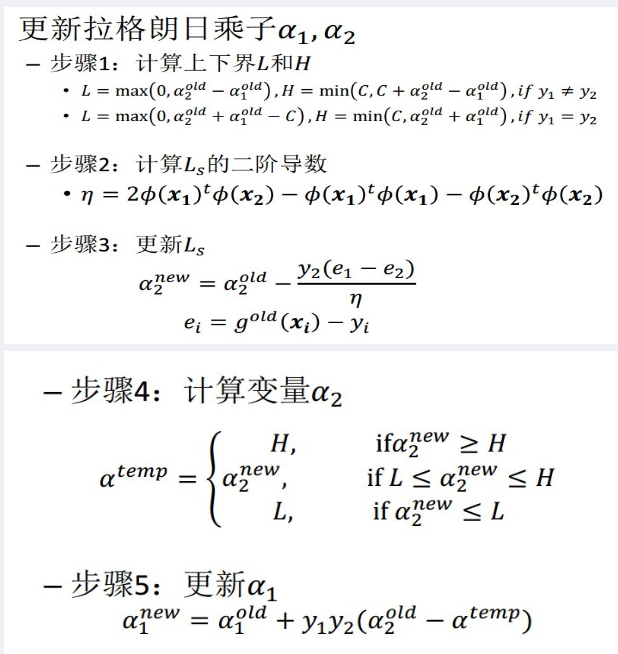
对L分别求偏导，并令其为0, 代入原式化简之后得到和原来一样的目标函数：



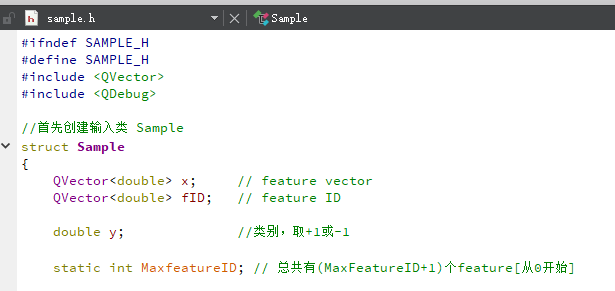
以上讨论的都是在线性可分情况进行讨论的，但是实际问题中给出的数据并不是都是线性可分的。对于非线性的情况，SVM 的处理方法是选择一个核函数 κ(⋅,⋅) ，通过将数据映射到高维空间，来解决在原始空间中线性不可分的问题。在线性不可分的情况下，支持向量机首先在低维空间中完成计算，然后通过核函数将输入空间映射到高维特征空间，最终在高维特征空间中构造出最优分离超平面，从而把平面上本身不好分的非线性数据分开。本程序采用的是Gaussian核函数。

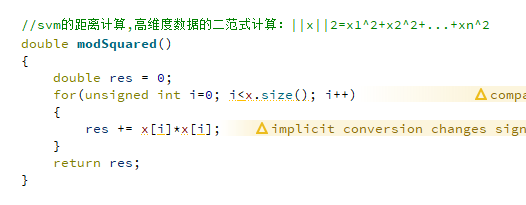
1. **实现基于SMO算法的SVM**

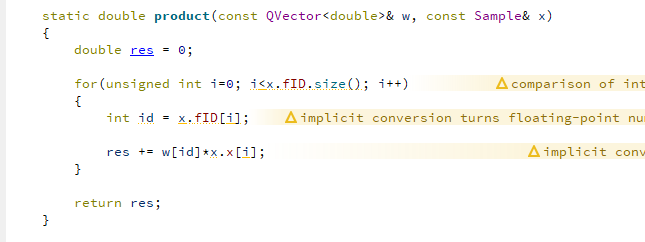
SMO算法的基本思想是将Vapnik在1982年提出的Chunking方法推到极致，SMO算法每次迭代只选出两个分量ai和aj进行调整，其它分量则保持固定不变，在得到解ai和aj之后，再用ai和aj改进其它分量。与通常的分解算法比较，尽管它可能需要更多的迭代次数，但每次迭代的计算量比较小，所以该算法表现出较好的快速收敛性，且不需要存储核矩阵，也没有矩阵运算。算法原理可参考下图：



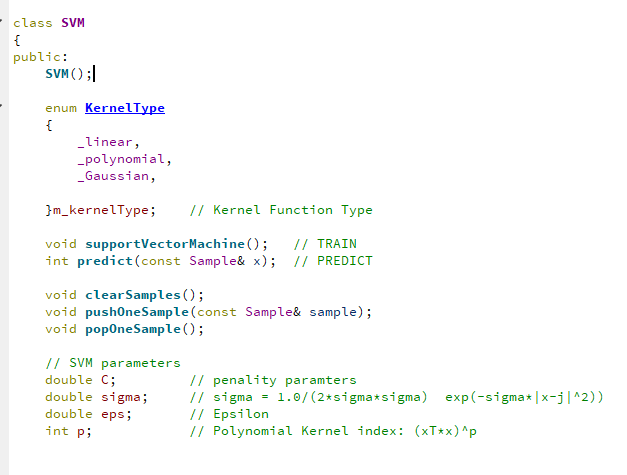
1. **C++的实现（QT5.11.3、Win10）**
2. 样本点的实现（具体实现在Sample.h中）







1. 构造SVM类

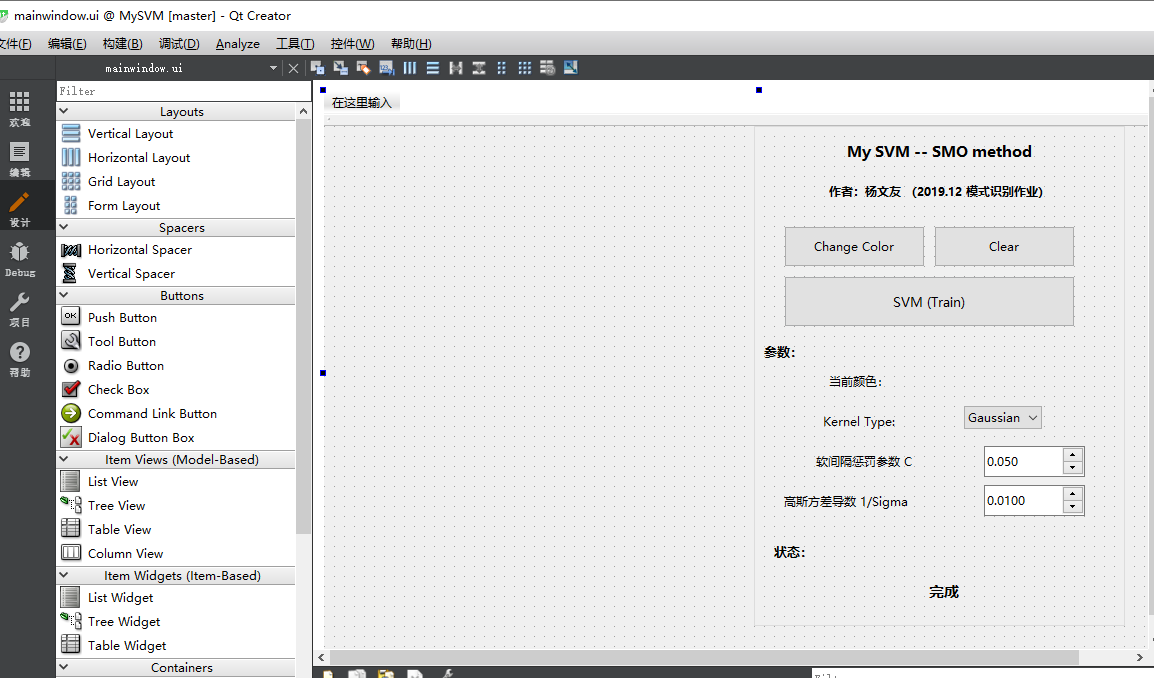


相关核函数的实现：



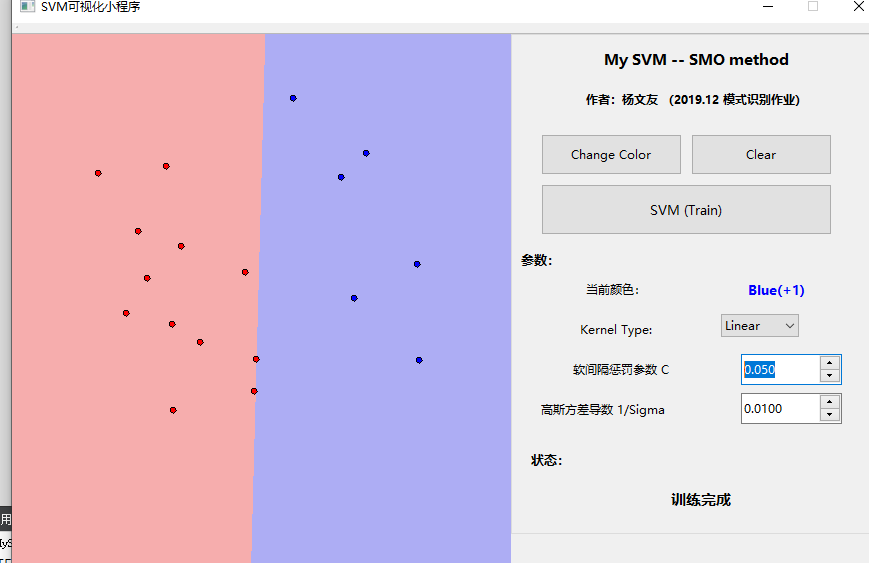


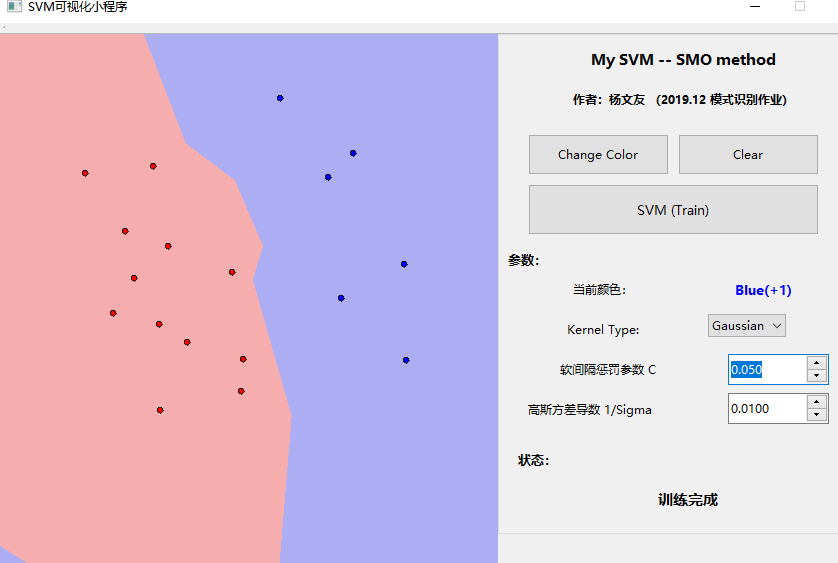
界面主体设计来源于网上的相关资源，修改了部分内容。



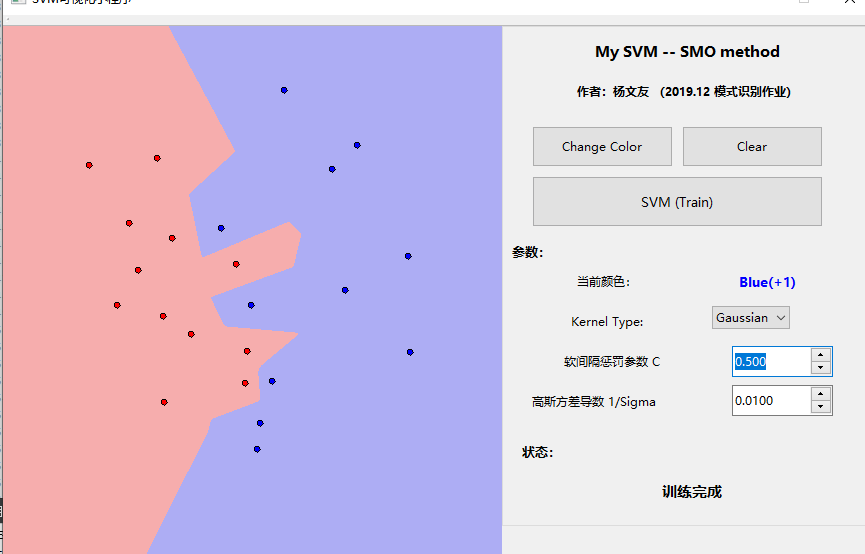
1. **运行截图**

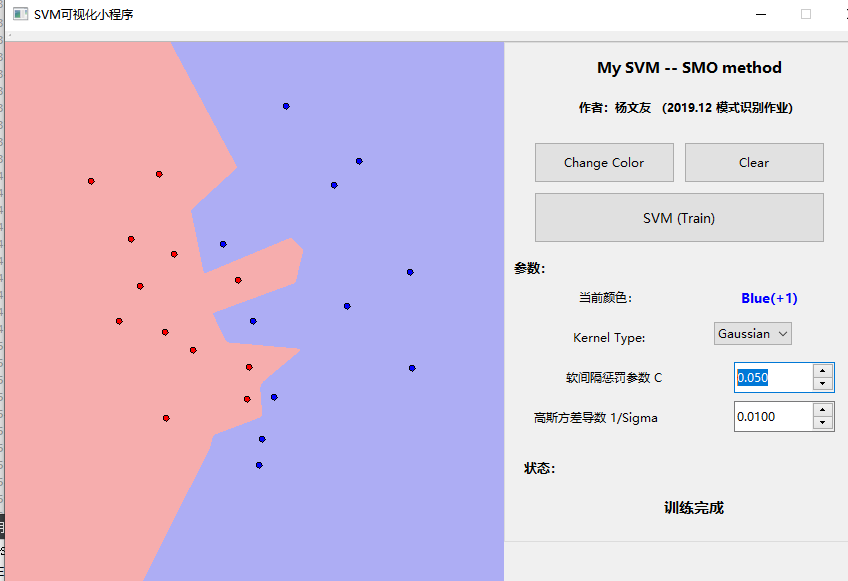
不同核函数（线性、Gaussian）的训练结果



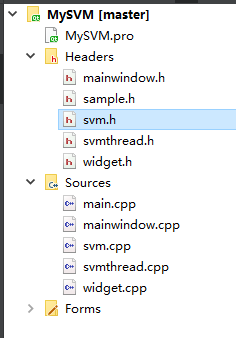


不同软间隔的效果：





**附：文件说明**

Headers为有关头文件，样本类的实现在sample.h里；svm的实现在svm.h里；训练过程在svmthread.h;文件夹Forms下图形界面素材。