

kNN 和 SVM 进行手写数字识别

学生姓名：沈之豪

院系名称：计算机学院

所在班级：计算机与科学 8 班

学生学号：2018302080181

指导教师：彭敏

摘要

当今时代，进行图像处理最热门的方法就是套用深度学习中的 CNN 模型。然而，传统的 CNN 模型只有在规模较大（网络层数多，数据量大）的情况下才能拥有较好的效果。在数据集较小的情况下，其性能可能还不如一些精心设计的传统机器学习算法，如 SVM。本文尝试着从最简单的 kNN，再到最后使用强大的 SVM，来探究传统机器学习算法在手写数字识别上的表现。

关键字：kNN；SVM；SMO

目录

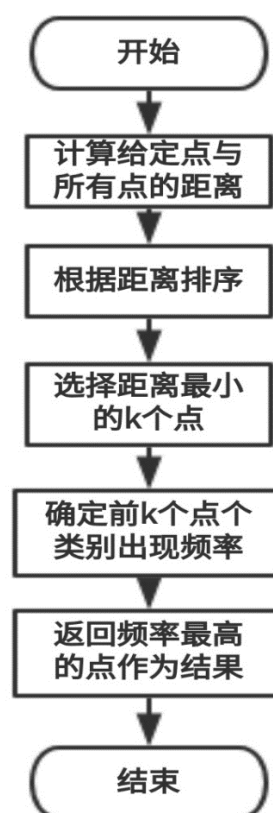
1. 使用 kNN 做手写数字识别	3
1.1 kNN 基本思想	3
1.2 数据预处理	4
1.3 数据集描述	5
1.4 Image2Vector	5
1.5 误差分析和参数调整	6
2. 使用 SVM 来做手写数字识别	7
2.1 SVM 基本概念	7
2.2 目标函数和约束	8
2.3 SMO 算法	13
2.4 核函数	19
2.5 多分类问题	20
2.6 误差分析与参数调试	21
3. 总结	21
3.1 优点和缺点	21
3.2 kNN 和 SVM 之间的比较	22
3.3 TODO list	23

1. 使用 kNN 做手写数字识别

1.1 kNN 基本思想

首先我们要知道 kNN 是用来处理多分类问题的。当给定一个输入的时候，相应的算法必须得返回一个确定的分类结果。那算法是怎么来得到分类的呢？简单地来说就是选择 k 个与该输入欧式距离最近的样本，然后这 k 个样本中拥有最多样本的类别就是 kNN 的分类结果。

流程图如下所示：



1.2 数据预处理

在做统计任务的时候一个非常重要的点就是进行数据预处理。做 kNN 时这一点也是尤为重要。我们先来观察 n 维空间下欧式距离的表达式：

$$d = \sqrt{\sum_{i=1}^n d_i^2}$$

其中， d_i 是两个样本点对应特征的差。

从中我们不难看出，极端 d_i 对我们结果的影响很大。为了避免这一点，我们选择在处理数据之前对其进行归一化。所谓的归一化就是：

$$x_{new} = \frac{x - \min(x)}{\max(x) - \min(x)}$$

这样我们原来的数据集就以类似比率的形式存在了，我们就再也不用担心极端值的问题了。

```
# Normalize our data set
def AutoNorm(data_set):
    # Get all feature's limits
    min_val = data_set.min(0)
    max_val = data_set.max(0)
    val_range = max_val - min_val
    # Normalize our data set
    m = data_set.shape[0]
    normal_set = data_set - np.tile(min_val, (m, 1))
    normal_set = normal_set / np.tile(val_range, (m, 1))
    return normal_set, min_val, val_range
```

1.3 数据集描述

本次手写数字识别使用的数据集被分为两个部分——训练集和测试集。

识别的数字范围是 0~9。对应的，我就有许多张 0~9 的图片放在训练集文件夹和测试集文件夹里。

众所周知，分类问题是一种监督式的学习，因此我们也需要对应的数字标签。这些数字的标签都隐藏在我图片的文件名里面。

(P. S. 所有的图片都是已经经过二值化处理的。因为根据灰度二值化图像不是我们本次的重点，因此我也没有放在代码里面)

根据以上的描述，如果我要把训练集或者测试集中的所有样本都导入到我的程序中，我应该要做以下两步：

1. 找到目录，获取目录中全部文件。
2. 将每一个文本文件的内容当作特征，文件名的一部分当作标签。

这样一来我们就顺利地对我们的数据导入了。

1.4 Image2Vector

还有一点要注意的就是我数字图像的像素是 $32 * 32$ ，这是一个二维的数据。传统的 kNN 算法只能处理一维的特征量。为了能和传统的 kNN 算法顺利接轨，我们得将这 $32 * 32$ 个数据转化为向量。

最简的做法就是不断地堆叠，比如第 i 行第 j 列的像素，就位于我向量的 $32 * i + j$ 个索引上。

```

# Convert matrix to vector
def Img2Vector(filename):
    # Return matrix
    ret_val = np.zeros((1, 1024))
    # Read file's content
    file = open(filename)
    content = file.readlines()
    for i in range(32):
        line = content[i]
        for j in range(32):
            ret_val[0, 32 * i + j] = int(line[j])
    return ret_val

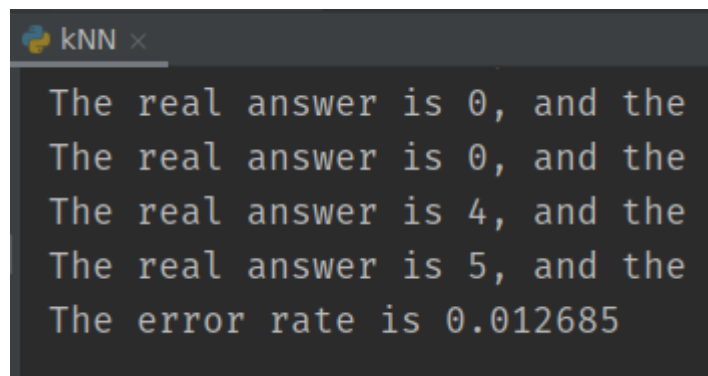
```

1.5 误差分析和参数调整

kNN 是一种特殊的机器学习的算法，它特殊就特殊在它不需要训练集的训练。同时，作为代价，每一次运行的时候都得加载整个数据集。

其它模型都要根据训练集上的误差来对参数进行调整。可惜 kNN 既不需要训练集，也没有参数可训练。

kNN 最终的结果如下所示：



```

kNN x
The real answer is 0, and the
The real answer is 0, and the
The real answer is 4, and the
The real answer is 5, and the
The error rate is 0.012685

```

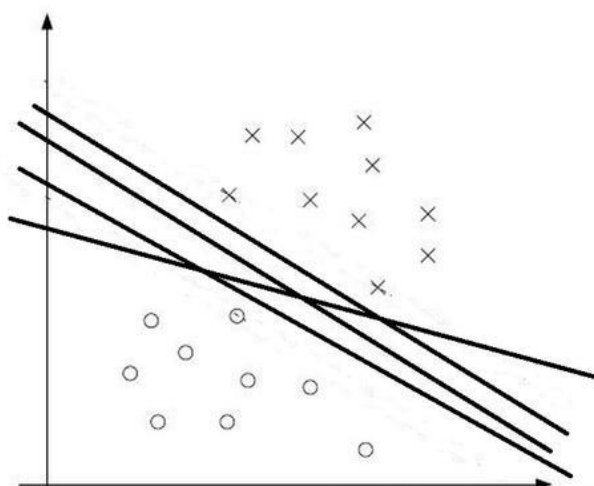
2. 使用 SVM 来做手写数字识别

因为使用 SVM 来做手写数字识别这个任务前人已经做得非常好了，所以本文的重点主要会放在有关 SVM 的推导上。

2.1 SVM 基本概念

这里简单的介绍一下 SVM 里面的一些基本的概念。

SVM 一开始是专门用来求解二分类问题的。以仅有两个特征的二分类问题为例，如果这个数据恰好能被一条直线分隔，如下图所示，那就称它是线性可分的。分隔它的直线就叫做超平面。



注意到如果考虑更多特征，那么特征空间的维度将会上升，原来的超平面就会从直线变成平面，再从平面变成一个立体。总之，他都满足下面这个式子，满足下面这个式子的，也都被称作超平面：

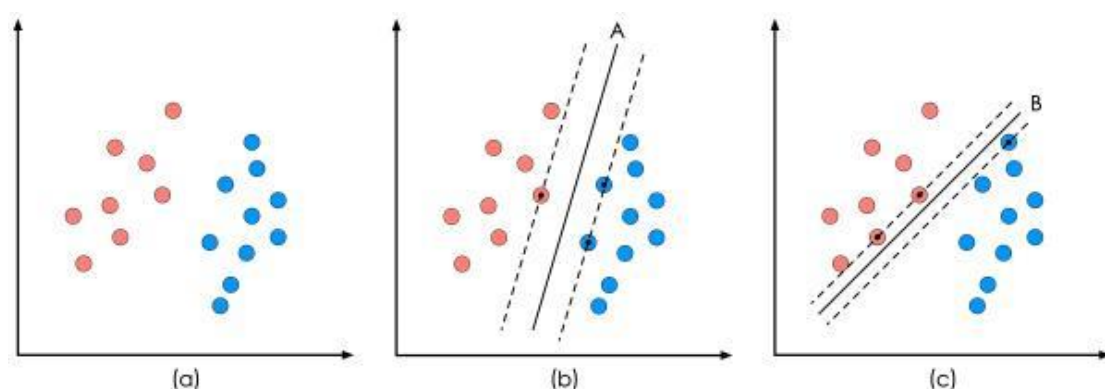
$$\mathbf{w}^T \mathbf{x} + b = 0$$

N 维空间里满足这个约束的对象，其维度将是 N-1 维的。正如在

2 维平面里超平面是一条直线，3 维立体空间里超平面将会是一个平面。在这个 N 维空间里，如果超平面也能做出正确的分类，我们就称这个数据集是线性可分的。这样我们就把目光从二维投向了高维空间。

不过，正如图上所示的那样，超平面可能会有多个。SVM 认为，只有选择最中间的那条直线，训练出来的分类器才会有更强的泛化效果。这里所谓“最中间”说得太模糊了，用更加严谨的说法来说，就是让所谓的支持向量距离我们的超平面最远。支持向量就是离超平面最近的点。所以 SVM 最基本的思想就是让最小距离最大化。

下图中，(b)所确定的超平面就比(c)所确定的超平面要优秀。



2.2 目标函数和约束

2.2.1 直观的目标函数与约束

之前我们提到过让最小距离最大化，那么我们一定得量化距离。

N 维空间中的点到超平面的欧式距离为：

$$d = \frac{|\mathbf{w}^T \mathbf{x}_i + b|}{\|\mathbf{w}\|}$$

当然我们的超平面也得满足一定的条件，那就是得把我们所有的点都

正确分类，这个就是我们的约束：

$$\begin{cases} \mathbf{w}^T \mathbf{x}_i + b \geq 0 & \forall y_i = 1 \\ \mathbf{w}^T \mathbf{x}_i + b \leq 0 & \forall y_i = -1 \end{cases}$$

注意我们这里把二分类量化为-1 和 1，仅仅是为了计算上的方便。上述的分类想法是我们最直接的分类想法，总结一下就是：

$$\begin{aligned} & \max_{\mathbf{w}, b} \frac{|\mathbf{w}^T \mathbf{x}_i + b|}{\|\mathbf{w}\|} \\ \text{s.t. } & \begin{cases} \mathbf{w}^T \mathbf{x}_i + b \geq 0 & \forall y_i = 1 \\ \mathbf{w}^T \mathbf{x}_i + b \leq 0 & \forall y_i = -1 \end{cases} \end{aligned}$$

其中 \mathbf{x}_i 是对应支持向量。

2.2.2 转化后的目标函数与约束

上述想法太过简单，直接导致了计算上的复杂性，所以我们得想些办法来尽可能简化我们的计算。

之前我们用到 $\mathbf{w}^T \mathbf{x}_i + b > 0$ ，这个条件过于宽泛了，其实我们完全可以引入此时支持向量到超平面的距离 d ，此时我们的约束条件将变为：

$$\begin{cases} \mathbf{w}^T \mathbf{x}_i + b \geq d & \forall y_i = 1 \\ \mathbf{w}^T \mathbf{x}_i + b \leq -d & \forall y_i = -1 \end{cases}$$

将这个式子稍作转化，记：

$$\mathbf{w}'^T = \frac{\mathbf{w}^T}{d}, b' = \frac{b}{d}$$

我们就得到了：

$$\begin{cases} \mathbf{w}'^T \mathbf{x}_i + b' \geq 1 & \forall y_i = 1 \\ \mathbf{w}'^T \mathbf{x}_i + b' \leq -1 & \forall y_i = -1 \end{cases}$$

这个不等式组可以简化为：

$$y_i(\mathbf{w}'^T \mathbf{x}_i + b') \geq 1$$

方便起见，我们之后仍然用 \mathbf{w} 和 b ，而不是使用这里新的 \mathbf{w}' 和 b' 。

虽然我们的方程改变了，但是我们的距离公式还是没有变。他依然是：

$$\max_{\mathbf{w}, b} \frac{|\mathbf{w}^T \mathbf{x}_i + b|}{\|\mathbf{w}\|}$$

值得注意的是，原来约束的不等式组等号成立的条件为： \mathbf{x}_i 本身就是支持向量。经过转化后这一点并没有改变，也就是说，我们有：

$$\mathbf{w}^T \mathbf{x}_i + b = 1$$

当且仅当 \mathbf{x}_i 为支持向量的时候。我们的目标方程和约束条件就变为：

$$\max_{\mathbf{w}} \frac{1}{\|\mathbf{w}\|}$$

$$s.t. \quad y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1$$

稍微改动一下变为我们熟悉的形式，更有利于后面的计算：

$$\min_{\mathbf{w}} \frac{1}{2} \|\mathbf{w}\|^2$$

$$s.t. \quad y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1$$

最后再来考虑一点：要知道不是所有的数据集都是 100%线性可分的。也就是说，会存在一个分错的情况，但是少量的分错我们实可以容忍的。为了容忍这个少量分错的情况，我们要引入一个松弛因子和惩罚系数。

松弛因子的想法是，约束条件 $y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1$ 过于严苛，未能

正确分类的肯定达不到这一个要求，也就是说，对于未正确分类的点，左边的式子是无法大于等于 1 的。他肯定大于等于一个比 1 小的数，我们不妨设：

$$y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 - \theta_i$$

$$\theta_i \geq 0$$

这样通过引入一个新的变量 θ ，我们就能够模拟那些所谓的离群点了。当然现在离群点只在约束里面，显然只有约束还是不够的，如果不对离群点做出惩罚的话，我们可以拥有无数个超平面。因此我们得把他考虑进我们的目标函数里面，具体的做法是引入惩罚项，这样一来我们就得到了：

$$\begin{aligned} \min_{\mathbf{w}} \quad & \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^m \theta_i \\ \text{s.t.} \quad & y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 - \theta_i \\ & \theta_i \geq 0 \end{aligned}$$

其中 C 就是我们的惩罚因子。

2.2.3 利用 KKT 转化为对偶问题

求解上述带约束的优化问题，最麻烦的就是那个约束，这里我们采用 KKT 条件来将其转化为对偶问题。

首先我们引入拉格朗日乘子，得到拉格朗日函数：

$$L(\mathbf{w}, \mathbf{b}, \boldsymbol{\theta}) = \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^m \theta_i - \sum_{i=1}^m a_i (y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1 + \theta_i) - \sum_{i=1}^m b_i \theta_i$$

KKT 条件要求我们：

$$\begin{cases} \frac{\partial L}{\partial \mathbf{w}} = 0, \frac{\partial L}{\partial b} = 0, \frac{\partial L}{\partial \boldsymbol{\theta}} = 0 \\ a_i (y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1 + \theta_i) = 0 \\ a_i > 0, b_i > 0 \end{cases}$$

我们可以得到：

$$\frac{\partial L}{\partial \mathbf{w}} = 0 \Rightarrow \mathbf{w} = \sum_{i=1}^m a_i y_i \mathbf{x}_i$$

$$\frac{\partial L}{\partial b} = 0 \Rightarrow \sum_{i=1}^m a_i y_i = 0$$

$$\frac{\partial L}{\partial \boldsymbol{\theta}} = 0 \Rightarrow a_i = C - b_i$$

看来我们可以利用 a 来计算 \mathbf{w} 了，毕竟 y_i 和 \mathbf{x}_i 都是已知量。

又已知 $b_i \geq 0$ ，我们可以得到：

$$a_i \leq C$$

将以上式子带入原来的式子，化简，我们就得到了新的带约束函数的公式：

$$\max_a \sum_{i=1}^m a_i - \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m a_i a_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

$$s. t. C \geq a_i \geq 0 \quad \forall i \in N^+$$

$$\sum_{i=1}^m a_i y_i = 0$$

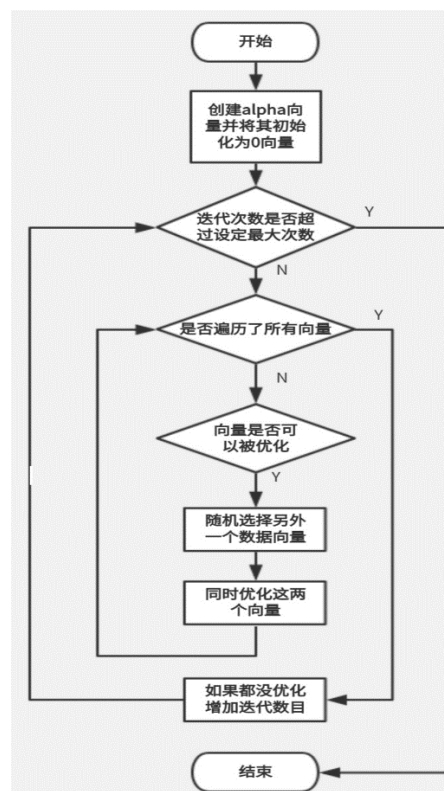
我们的任务就是设计一个算法来求解上述任务，为此我打算使用 SMO 算法。

2.3 SMO 算法

2.3.1 SMO 算法基本思想

其实之前获得的还是一个带约束的优化问题。SMO 算法的基本思想是，为了满足约束条件，我先算出每一个 a 的范围，然后通过费马引理计算原来的拉格朗日函数，得到 a 的最优解。不过这个最优解还是要和之前计算出来的 a 相比较，如果在这个范围内那最好。如果不再这个范围内，那就只能用范围边界来代替了。

流程图如下图所示：



2.3.2 选取合适的 a

之前在流程图中你们也看到了，那就是我们要挑选两个 a 出来，之所以这么做，是因为我们要满足所有 a 与 y 乘积的和要等于 0 这一个条件。如果我每次只更新一个 a ，那么这个条件显然不会满足。因此我要挑两个 a 出来——当然了，这两个 a 不能随便挑。已经优化完毕的 a 就没有什么要更新的必要。

$$\text{记 } u_i = \mathbf{w}^T \mathbf{x} + b$$

根据 SVM 的理论，我们有：

$$a_i = 0 \Leftrightarrow y_i u_i \geq 1$$

$$0 < a_i < C \Leftrightarrow y_i u_i = 1$$

$$a_i < 0 \Leftrightarrow y_i u_i \leq 1$$

这三种情况分别对应着：

1. 最普通的，非支持向量，也非离群的点。
2. 支持向量。
3. 离群点。

一旦 a_i 和 $y_i u_i$ 的关系出现了矛盾，我们就要更新这些 a_i 。

2.3.3 更新两个权重

获取 a 的上下限

首先我们要获取每一次变化的上下限。最重要的一点，就是所有 a 和对应 y 乘积的和不会变。如果我们取出的两个 a 分别是 a_1 和 a_2 ，

那么他们一定满足如下关系：

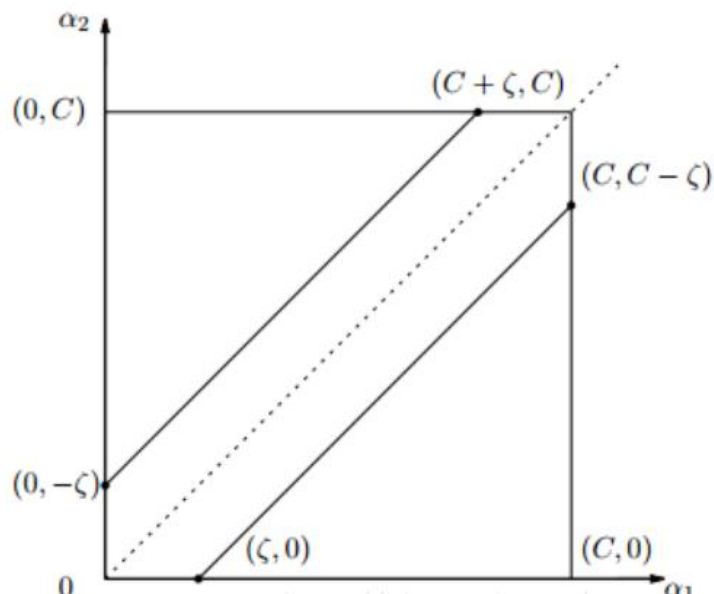
$$y_1 \cdot a_1 + y_2 \cdot a_2 = \vartheta$$

其中 ϑ 是一个常数，不管 a_1 和 a_2 怎么更新，这一个等式都是不会变的。

上一节中提到我们的 a 必须在合适的范围之内，现在又有了这个等式。两个条件结合在一起，我们不妨将其可视化，看得更清楚一点：

我们以 $y_1 = 1$, $y_2 = -1$ 为例，此时我们有：

$$a_2 = a_1 - \vartheta$$



对这幅图稍微做一点解释：

1. 首先在第一象限的那个大的矩形，就是我们上一节中得到的 a 的范围： $[0, C]$
2. 矩形的中的两条直线对应了 ϑ 取不同值的两种情况，为了与矩形取得交集，我们的 ϑ 只有两种范围：
 - a) $\vartheta \in [0, C]$ ，这是图中的位于下方的一条直线。

b) $\vartheta \in [-C, 0]$, 这是图中位于上方的一条直线。

3. 实际上我们 a 的取值范围就是矩形和我们直线的交集。这个交集正是我们矩形中的线段部分。

对于这种情况, 我们 a_2 的取值范围是:

$$a_2 \in \begin{cases} [0, C - \vartheta], & \vartheta \in [0, C] \\ [-\vartheta, C], & \vartheta \in [-C, 0] \end{cases}$$

我们可以将这个复杂的式子合一化, 即得到 $a_2 \in [L, H]$ 的样式。此时:

$$L = \max(0, -\vartheta)$$

$$H = \min(C, C - \vartheta)$$

这下子式子就很优雅了, 我们考虑一下当 y_1 和 y_2 取别的值的情况, 特别的, 由于之前考虑了 $y_1 = 1, y_2 = -1$ 的情况, 这里我们来考虑 $y_1 = -1, y_2 = 1$ 的情况。不过我们并不需要进行重复的分析, 只要稍加观察即可; 注意到:

$$a_2 = a_1 + \vartheta$$

所以只要把所有的 $-\vartheta$ 都改为 ϑ 即可。我们可以直接写出结果:

$$L = \max(0, \vartheta)$$

$$H = \min(C, C + \vartheta)$$

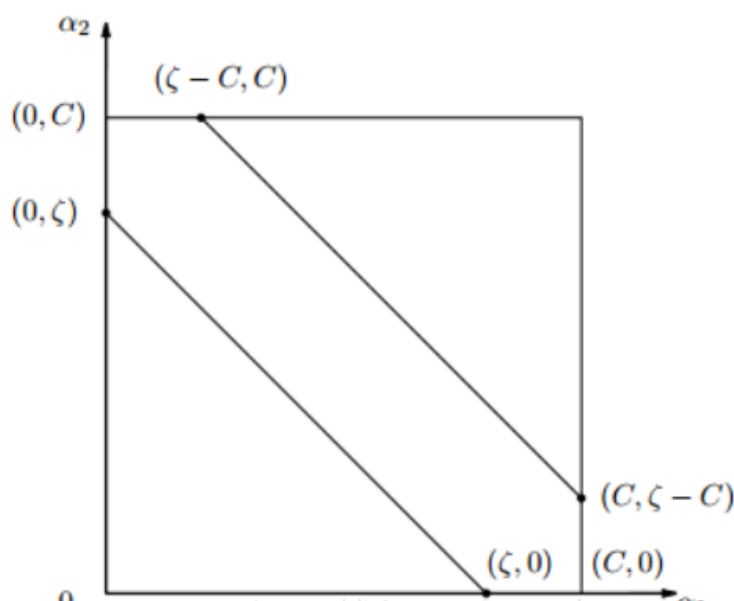
细心观察, 我们可以把 y_1 和 y_2 异号的情况综合起来考虑, 只要把 ϑ 修改为 $a_2 - a_1$ 即可。这样我们就得到了:

$$L = \max(0, a_2 - a_1)$$

$$H = \min(C, C + a_2 - a_1)$$

对于 y_1 和 y_2 同号的情况, 我们也可以做这样的考虑, y_1 和 y_2 同号

时图像如下所示：



最终我们可以得到：

$$L = \max(0, a_2 + a_1 - C)$$

$$H = \min(C, a_2 + a_1)$$

写在一起就是：

$$\begin{cases} L = \max(0, a_2 - a_1), H = \min(C, C + a_2 - a_1) & \text{when } y_1 \neq y_2 \\ L = \max(0, a_2 + a_1 - C), H = \min(C, a_2 + a_1) & \text{when } y_1 = y_2 \end{cases}$$

利用导数求 a2 最优解

之前我们已经求得了 a_2 的范围。这次我们先抛开范围来求解关于 a_2 的最优解。这个其实没什么好讲的，联立以下式子就可以得出

来：

$$w(a_2) = \sum_{i=1}^n a_i - \frac{1}{2} \sum_{i,j} y_i y_j x_i^T x_j a_i a_j$$

$$a_1 y_1 + a_2 y_2 = \vartheta$$

这里直接给出答案，因为结果虽然简单但并不直观：

记：

$$E_i = f(x_i) - y_i$$

$$\eta = x_1^T x_1 + x_2^T x_2 - 2x_1^T x_2$$

得到：

$$a_2 = a_2 + \frac{y_2(E_1 - E_2)}{\eta}$$

结合前两部获得最终的结果

记更新后的 a_2 为 a_{2new} ，则：

$$a_2 = \begin{cases} H & a_2^{new} \geq H \\ a_2^{new} & L < a_2^{new} < H \\ L & a_2^{new} \leq L \end{cases}$$

利用之前提到的 a_1 和 a_2 的关系我们又可以得出

$$a_1 = a_1^{old} + y_1 y_2 (a_2^{old} - a_2)$$

由此可见再更新 a_1 和 a_2 的时候我们还需要保存下 a_1 和 a_2 原来的值。

2.3.4 更新 b

我们需要根据 a 的取值范围，去更新 b 的值，来使得间隔最大化，首先对于 b 的更新，我们只能回到之前的约束条件，这里的 x_1 代表的是支持向量，所以不用考虑松弛问题：

$$y_2(w^T x_1 + b) = 1$$

两边都乘以 y_1 得：

$$\sum_{i=1}^m a_i y_i x_i x_1 + b = y_1$$

这个时候我们就可以把之前得到得 a_2 和 a_1 分离出来了，得到：

$$b_1^{new} = b^{old} - E_1 - y_1(a_1^{new} - a_1^{old})x_1^T x_1 - y_2(a_2^{new} - a_2^{old})x_2^T x_2$$

同理我们可以得到另外一个 b ：

$$b_2^{new} = b^{old} - E_2 - y_1(a_1^{new} - a_1^{old})x_1^T x_2 - y_2(a_2^{new} - a_2^{old})x_2^T x_2$$

当 b_1 和 b_2 都是有效的时候，他们是相等的，即：

$$b^{new} = b_1^{new} = b_2^{new}$$

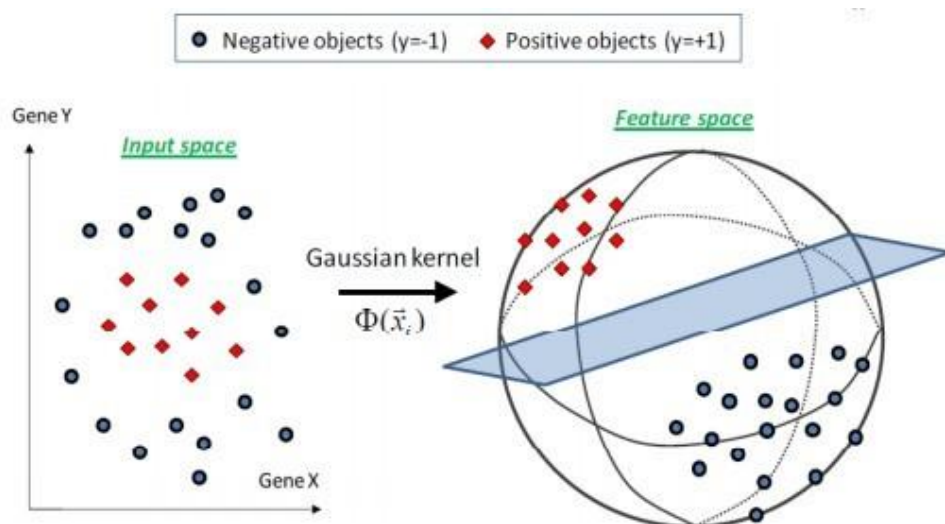
当这个条件不满足的时候，SMO 选择他们的中点作为新的阈值：

$$b = \begin{cases} b_1 & 0 < a_1^{new} < C \\ b_2 & 0 < a_2^{new} < C \\ \frac{b_1 + b_2}{2} & otherwise \end{cases}$$

为此，我们把 SMO 蕴含的数学知识也都全部推导了一遍。

2.4 核函数

之前我们已经提到过，SVM 要求我们的数据集必须是线性可分的。如果不是 100%线性可分的，我们就得引入松弛变量。但是不管怎样，我们只能用 $w^T x + b = 0$ 这种超平面来分隔我们的数据，然而，现实生活中的很多东西，并不都是线性可分的。为了让我们的 SVM 能够处理非线性的问题，我们就得引入核函数。



所谓的核函数就是把我们原来的特征映射到新的高维的特征空间中去，把低维的，线性不可分的问题，转化为高维的，线性可分的问题。

直接对所有特征进行维度的转换可能会遇到维度爆炸的情况。如果支持 SVM 的求解只用到内积运算，而在低维输入空间又存在某个函数 $K(x, x')$ ，它恰好等于在高维空间中这个内积，即

$$K(x, x') = \langle \varphi(x) \cdot \varphi(x') \rangle$$

那么 SVM 就不用计算复杂的非线性变换，而由这个函数 $K(x, x')$ 直接得到非线性变换的内积，使大大简化了计算。这样的函数 $K(x, x')$ 称为核函数。我们只要把我们原来代码中有内积的地方都改为这个核函数就行了。

2.5 多分类问题

SVM 的问题本身是二分类的问题，但是我们的手写数字识别要求我们能够识别 $0 \sim 9$ 。这里我打算使用 OVR 算法。就是同时建立 $0 \sim 9$ 这

10 个数字的分类器，通过观察这 10 种情况的分类结果来做出最后的判断。这个本质上就是一个模型集成的问题。

2.6 误差分析与参数调试

SVM 需要调试的参数共有以下不等：

1. 松弛变量的乘法系数 C 。
2. 核函数的种类及对应的参数。

由于选择的数据集已经经过了数据清洗，所以我们可以看到分类的效果特别好，不管是训练集还是测试机的误差都是 0%。

```
svm (1) x
The real answer is 4, and the classifier came back with 4
The real answer is 5, and the classifier came back with 5
The training set error is 0.000000
```

```
svm (1) x
The real answer is 0, and the classifier came back with 0
The real answer is 0, and the classifier came back with 0
The real answer is 4, and the classifier came back with 4
The real answer is 5, and the classifier came back with 5
The test set error is 0.000000
```

3. 总结

3.1 优点和缺点

我主要想从数据方面来说一下本文的优点和缺点

1. 优点：数据清洗做得非常好。类别内部的样本分布很均匀。

不同类别间也没有出现样本不均衡的现象出现。

2. 缺点：数据清洗做得太好了。以至于我们无法确定对于手写数字识别任务中最好的参数。

3.2 kNN 和 SVM 之间的比较

在相同的数据集上，我们可以看到不同的模型表现的优劣性也完全不同。就以本文中使用到的两个算法模型为例，SVM 表现得就完全比 kNN 要出色。

究其原因，只能说 SVM 是一种精心设计的，无可挑剔的算法；而 kNN 仅仅是考虑在特征空间中的欧式距离。实际上，如果仅仅是考虑欧式距离的话，其实是把每一个特征都等同看待，这是不对的，正如其它分类例子中不同的特征有不同的权重一样，手写数字处理不同的特征也拥有着不同的权重。所以说 kNN 算法还有很大改进的空间。

SVM 还有一个更加重要的优势——每次调试的时候不需要重新加载数据集。只要把每次训练的参数都保存在硬盘中，要用的时候拿出来就行。而 kNN 算法中少了一个训练的步骤，也就是说，对他而言训练集和测试集都一样，拿来就可以训练，为此付出的代价就是，每次调试的时候不得不重新加载数据集。显然当数据集过大的时候，我们就需要等待更长的时间，而且此时传统机器学习算法就不占优势，到时候肯定会采用深度神经网络的算法了。

3.3 TODO list

有了之前的这些经验，我们就应该让我们的算法落实到实际的应用中。结合之前的缺点，可以做的事情还有：

1. 获取更多的数据作为测试集。
2. 制作 GUI，尝试提供接口做数字识别方面的应用。
3. 本文主要探讨的是单个数字的情况。如果是多个数字的情况该怎么办？
4. 在图像二值化的过程中，采用了灰度图像。到最后应用的时候，获得的肯定是更为复杂的彩色图，这时候该怎么转换？
5. 数据集中默认的像素是 $32 * 32$ 。到时候接受的图像像素肯定不会固定在 $32 * 32$ ，甚至 $32 * 32$ 的像素中都放不下该数字。这时候又该怎么处理？