第七周学习笔记—支持向量机

2022-05-26

第一章　支持向量机

在监督学习中，许多学习算法的性能都非常类似，因此，重要的不是我们该选择使用学习算法 A 还是学习算法 B，而更重要的是，应用这些算法时，所创建的大量数据在应用这些算法时，表现情况通常依赖于我们的水平。比如：我们在为学习算法所设计的特征量的选择，以及如何选择正则化参数，诸如此类的事。

有一个更加强大的算法被广泛的应用于工业界和学术界，它被称为支持向量机 $(Support \quad Vector \quad Machine)$。

与逻辑回归和神经网络相比，支持向量机，或者简称 SVM，在学习复杂的非线性方程时提供了一种更为清晰，更加强大的方式。

1.1　优化目标

正如我们之前开发的学习算法，我们从优化目标开始。

为了描述支持向量机，事实上，将会从逻辑回归开始展示我们如何一点一点修改来得到本质上的支持向量机。

在逻辑回归中，我们已经知道了假设函数 $h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$ 以及 $Sigmoid$ 激活函数的图像。为了解释一些数学知识，我们将使用 $z = \theta^T x$。

现在考虑下我们想要逻辑回归做什么：

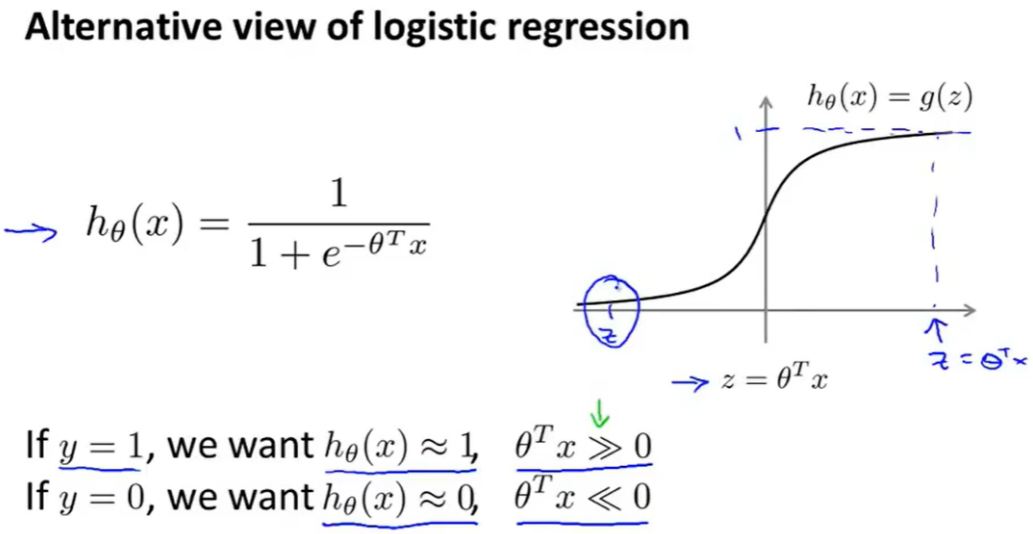
如果有一个 $y = 1$ 的样本，意思是不管是在训练集中或是在测试集中，又或者在交叉验证集中，总之是 $y = 1$，现在我们希望 $h_\theta(x)$ 趋近 1。

因为我们想要正确地将此样本分类，这就意味着当 $h_\theta(x)$ 趋近于 1 时，$\theta^T x$ 应当远大于 0，下图里的 $>>$ 意思是远远大于 0。

这是因为由于 z 表示 $\theta^T x$，当 z 远大于 0 时，即到了该图的右边，你不难发现此时逻辑回归的输出将趋近于 1。

相反地，如果我们有另一个样本，即 $y = 0$。我们希望假设函数的输出值将趋近于 0，这对应于 $\theta^T x$，或者就是 z 会远小于 0，因为对应的假设函数的输出值趋近 0。

我们用下图表示：



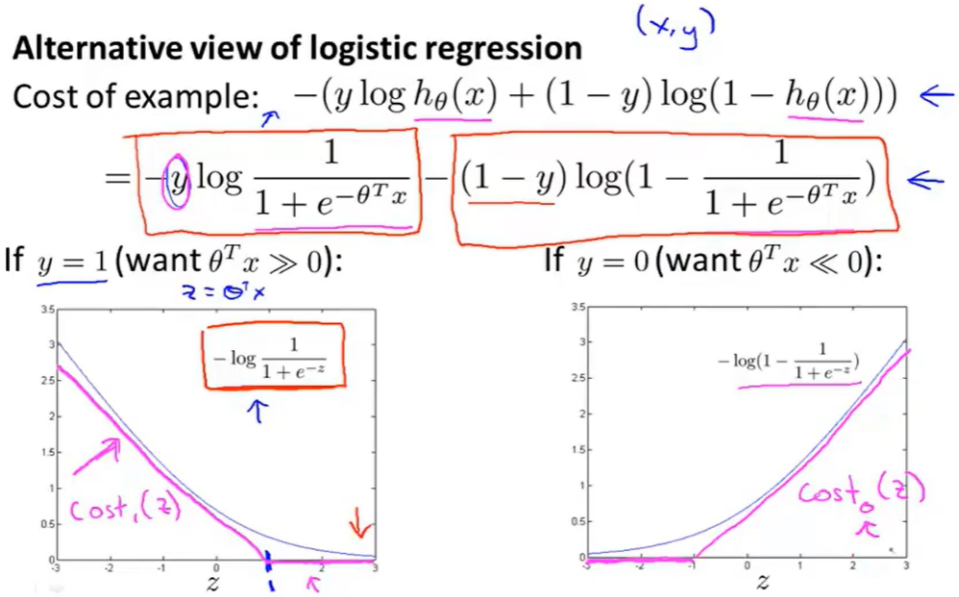
如果我们进一步观察逻辑回归的代价函数，不难发现每个样本 (x, y) 都会为总代价函数，增加一项，因此，对于总代价函数通常会有对所有的训练样本求和，并且这里还有一个 $\frac{1}{m}$ 项，但是，在逻辑回归中，这里的这一项就是表示一个训练样本所对应的表达式。现在，如果我将完整定义的假设函数代入这里。那么，我们就会得到每一个训练样本都影响这一项。

现在，先忽略 $\frac{1}{m}$ 这一项，但是这一项是影响整个总代价函数中的这一项的。

一起来考虑两种情况：

一种是 $y = 1$ 的情况；另一种是 $y = 0$ 的情况。

我们表现出不同取值情况下的图像，代价函数如下图所示：



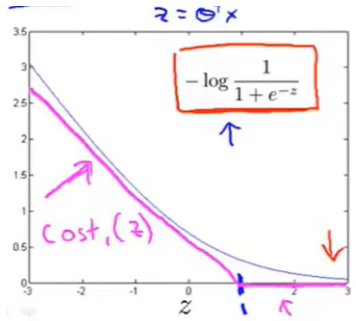
在第一种情况中，假设 $y=1$，此时在目标函数中只需有第一项起作用，因为 $y=1$ 时，$(1-y)$ 项将等于 0。因此，当在 $y=1$ 的样本中时，即在 (x,y) 中，我们得到 $y=-log(\frac{1}{1+e^{-z}})$ 这样一项。

如果画出关于 z 的函数，会看到左下角的这条曲线，我们同样可以看到，当 z 增大时，也就是相当于 $\theta^T x$ 增大时，z 对应的值会变的非常小。对整个代价函数而言，影响也非常小。

也就解释了，为什么逻辑回归在观察到正样本 $y=1$ 时，试图将 $\theta^T x$ 设置得非常大。因为，在代价函数中的这一项会变的非常小。

现在开始建立支持向量机，我们从这里开始：

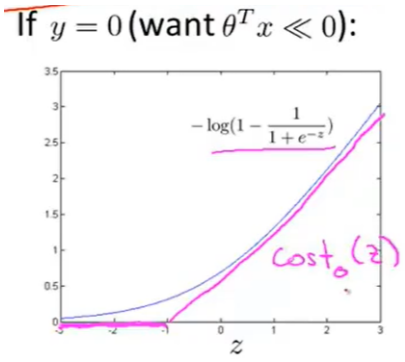
会上图右边这个代价函数开始，也就是 $-log(\frac{1}{1+e^{-z}})$ 一点一点修改，让我们取这里的 $z=1$ 点，我们画出将要使用的代价函数的图像如下：



新的代价函数将会水平的从这里到右边 (图外)，然后我们再画一条同逻辑回归非常相似的直线，但是，在这里是一条直线。那么，到了这里已经非常接近逻辑回归中使用的代价函数了。只是这里是由两条线段组成，即位于右边的水平部分和位于左边的直线部分，先别过多的考虑左边直线部分的斜率，这并不是很重要。但是，这里我们将使用的新的代价函数，是在 $y=1$ 的前提下的。

到这里也许能想到，这应该能做同逻辑回归中类似的事情。

目前，我们只是讨论了 $y=1$ 的情况，另外一种情况是当 $y=0$ 时，此时我们仔细观察代价函数只留下了第二项，因为第一项被消除了。如果当 $y=0$ 时，那么这一项也就是 0 了。所以上述表达式只留下了第二项。因此，这个样本的代价或是代价函数的贡献。将会由这一项表示。并且，如果你将这一项作为 z 的函数，那么，这里就会得到横轴 z。同样的，用我们上面学习到的方法替代。



如果我们用一个新的代价函数来代替，即这条从 0 点开始的水平直线，然后是一条斜线，像上图。那么，现在让我给这两个方程命名，左边的函数，我们称之为 $cost_1(z)$，同时，右边函数我们称它为 $cost_0(z)$。这里的下标是指在代价函数中，对应的 $y=1$ 和 $y=0$ 的情况，拥有了这些定义后，现在，我们就开始构建支持向量机。

对于支持向量机而言，实质上我们要将前后两部分分别替换为 $cost_1(z)$ 和 $cost_0(z)$，其中 $z=\theta^T x$。这里的代价函数就是之前所提到的那条线。代价函数的定义如下：

$$\min_\theta \frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)}\underbrace{\left(-\log h_\theta(x^{(i)})\right)}_{cost_1(\theta^T x^{(i)})} + (1-y^{(i)})\underbrace{\left(-\log(1-h_\theta(x^{(i)}))\right)}_{cost_0(\theta^T x^{(i)})}\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

因此，对于支持向量机，我们得到了这里的最小化问题。

然后，再加上正则化参数。现在，按照支持向量机的惯例，事实上，我们的书写会稍微有些不同，代价函数的参数表示也会稍微有些不同。

首先，我们要除去 $\frac{1}{m}$ 这一项，当然，这仅仅是由于人们使用支持向量机时，对比于逻辑回归而言，不同的习惯所致。

意思是：仅仅除去 $\frac{1}{m}$ 这一项，也会得出同样的 θ 最优值。因为 $\frac{1}{m}$ 这一项仅是个常量，因此，在这个最小化问题中，无论前面是否有 $\frac{1}{m}$ 这一项，最终我所得到的最优值 θ 都是一样的。

举一个样本为例：

假定有一最小化问题：即要求当 $(\mu-5)^2+1$ 取得最小值时的 μ 值，这时最小值为：$\mu=5$ 时取得最小值。

现在，如果我们想要将这个目标函数乘上常数 10，这里我的最小化问题就变成了：求使得 $10\times(\mu-5)^2+1$ 小的值 μ，然而，$\mu=5$ 时取得最小值。因此将一些常数乘以我们的最小化项，这并不会改变最小化该方程时得到 μ 值。

因此可以知道，仅仅除去 $\frac{1}{m}$ 这一项，也会得出同样的 θ 最优值。

于逻辑回归，在目标函数中，我们有两项：第一个是训练样本的代价，第二个是我

们的正则化项，我们不得不去用这一项来平衡。这就相当于我们想要最小化 A 加上正则化参数 λ，然后乘以其他项 B。这里的 A 表示这里的第一项，同时我用 B 表示第二项，但不包括 λ, 我们不是优化这里的 $A + \lambda \times B$。我们所做的是通过设置不同正则参数 λ 达到优化目的。这样，我们就能够权衡对应的项，是使得训练样本拟合的更好, 即最小化 A。还是保证正则参数足够小，也即是对于 B 项而言。

但对于支持向量机，按照惯例，我们将使用一个不同的参数替换这里使用的 λ 来权衡这两项。就是第一项和第二项我们依照惯例使用一个不同的参数称为 C，同时改为优化目标，$C \times A + B$。

因此，在逻辑回归中，如果给定 λ，一个非常大的值，意味着给予 B 更大的权重。而这里，就对应于将 C 定为非常小的值，那么，相应的将会给 B 比给 A 更大的权重。因此，这只是一种不同的方式来控制这种权衡或者一种不同的方法，即用参数来决定是更关心第一项的优化，还是更关心第二项的优化。

当然你也可以把这里的参数 C 考虑成 $\frac{1}{\lambda}$, 同 $\frac{1}{\lambda}$ 所扮演的角色相同，并且这两个方程或这两个表达式并不相同，因为 $C = \frac{1}{\lambda}$，但是也并不全是这样，如果当 $C = \frac{1}{\lambda}$ 时，这两个优化目标应当得到相同的值，相同的最优值 θ。

因此，这就得到了在支持向量机中我们的整个优化目标函数。然后最小化这个目标函数，得到 SVM 学习到的参数 C。

SVM hypothesis

$$\Rightarrow \min_{\theta} C\sum_{i=1}^{m}\left[y^{(i)}cost_1(\theta^T x^{(i)}) + (1-y^{(i)})cost_0(\theta^T x^{(i)})\right] + \frac{1}{2}\sum_{i=1}^{n}\theta_j^2$$

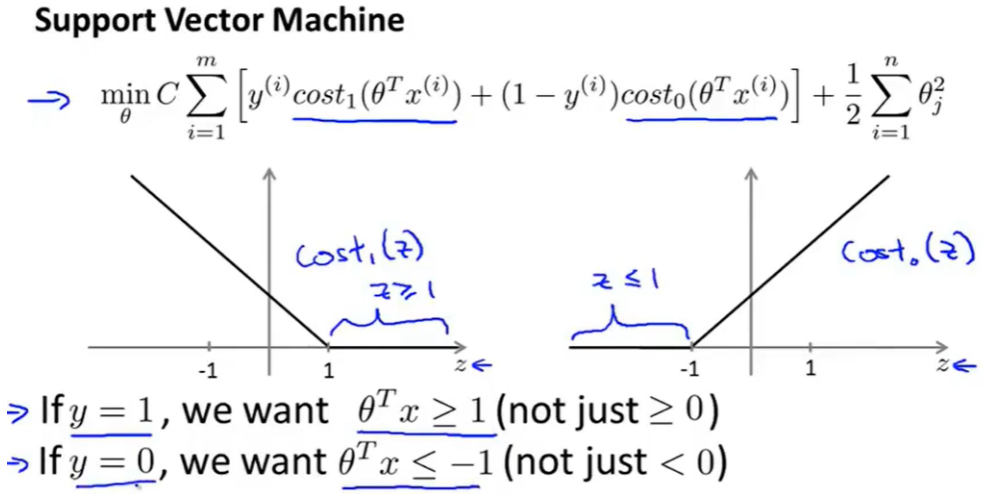
Hypothesis:

$$h_\theta(x) \begin{cases} 1 & \text{if } \theta^T x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

最后有别于逻辑回归输出的概率。在这里，我们的代价函数，当最小化代价函数，获得参数 θ 时，支持向量机所做的是它来直接预测 y 的值等于 1，还是等于 0。因此，这个假设函数会预测 1。当 $\theta^T x$ 大于或者等于 0 时，或者等于 0 时，所以学习参数 θ 就是支持向量机假设函数的形式。那么，这就是支持向量机数学上的定义。

1.2　大边界的直观理解

人们有时将支持向量机看作是大间距分类器。



这是我的支持向量机模型的代价函数，在左边这里我画出了关于 z 的代价函数 $cost_1(z)$，此函数用于正样本，而在右边这里我画出了关于 z 的代价函数 $cost_0(z)$, 横轴表示 z。

我们考虑一下，最小化这些代价函数的必要条件是什么?

如果你有一个正样本，$y=1$，则只有在 $z>=1$ 时，代价函数 $cost_1(z)$ 才等于 0。

换句话说，如果有一个正样本，我们会希望 $\theta^T x>=1$，反之，如果 $y=0$，我们观察一下，函数 $cost_0(z)$，它只有在 $z<=-1$ 的区间里函数值为 0。

事实上，如果你有一个正样本 $y=1$，则其实我们仅仅要求 $\theta^T x>=0$，就能将该样本恰当分出，这是因为如果 $\theta^T x>0$ 我们的模型代价函数值为 0，类似地，如果有一个负样本，则仅需要 $\theta^T x<=0$ 就会将负例正确分离。

但是，支持向量机的要求更高，不仅仅要能正确分开输入的样本，即不仅仅要求 $\theta^T x>0$，我们需要的是比 0 值大很多，比如大于等于 1，我也想这个比 0 小很多，比如我希望它小于等于-1，这就相当于在支持向量机中嵌入了一个额外的安全因子，或者说安全的间距因子。

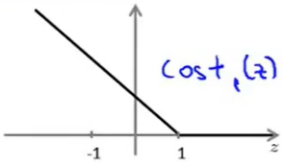
当然，逻辑回归做了类似的事情。但是让我们看一下，在支持向量机中，这个因子会导致什么结果。具体而言，我接下来会考虑一个特例。我们将这个常数 C 设置成一个非常大的值。比如我们假设 C 的值为 100000 或者其它非常大的数，然后来观察支持向量机会给出什么结果？

SVM Decision Boundary

$$\min_{\theta} C\sum_{i=1}^{m}\left[y^{(i)}cost_1(\theta^T x^{(i)}) + (1-y^{(i)})cost_0(\theta^T x^{(i)})\right] + \frac{1}{2}\sum_{i=1}^{n}\theta_j^2$$

Whenever $y^{(i)} = 1$:

$\theta^T x^{(i)} \geq 1$



Whenever $y^{(i)} = 0$:

$\theta^T x^{(i)} \leq 1$

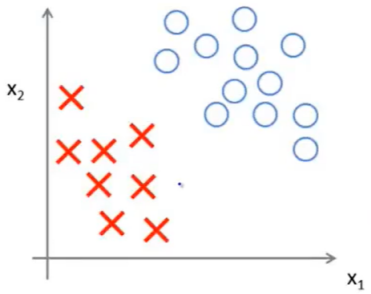
如果 C 非常大，则最小化代价函数的时候，我们将会很希望找到一个使第一项为 0 的最优解。因此，让我们尝试在代价项的第一项为 0 的情形下理解该优化问题。比如我们可以把 C 设置成了非常大的常数，这将给我们一些关于支持向量机模型的直观感受。

$$min_\theta C\sum_{i=1}^{m}[y^{(i)}cost_1(\theta^T x^{(i)}) + (1-y^{(i)})cost_0(\theta^T x^{(i)})] + \frac{1}{2}\sum_{j=1}^{n}\theta_j^2$$

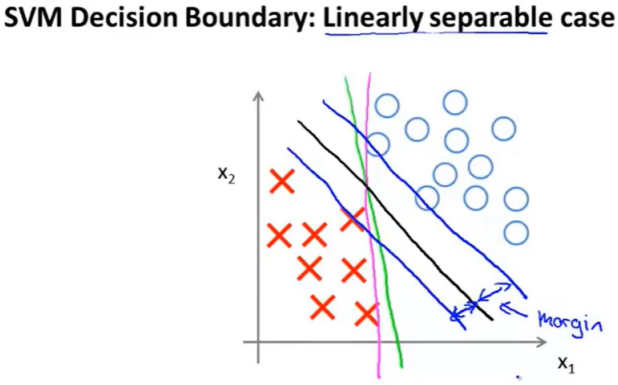
我们已经看到输入一个训练样本标签为 $y = 1$，你想令第一项为 0，你需要做的是找到一个 θ，使得 $\theta^T x >= 1$，类似地，对于一个训练样本，标签为 $y = 0$，为了使 $cost_0(z)$ 函数的值为 0，我们需要 $\theta^T x <= -1$。因此，现在考虑我们的优化问题。选择参数，使得第一项等于 0，就会导致下面的优化问题，因为我们将选择参数使第一项为 0，因此这个函数的第一项为 0，因此是 C 乘以 0 加上二分之一乘以第二项。这里第一项是 C 乘以 0，因此可以将其删去，因为我知道它是 0。

这将遵从以下的约束：$\theta^T x^{(i)} >= 1$，如果 $y^{(i)}$ 是等于 1 的，$\theta^T x^{(i)} <= -1$，如果样本 i 是一个负样本，这样当你求解这个优化问题的时候，当你最小化这个关于变量 θ 的函数的时候，你会得到一个非常有趣的决策边界。

SVM Decision Boundary: Linearly separable case



具体而言，如果你考察这样一个数据集，其中有正样本，也有负样本，可以看到这个数据集是线性可分的。我的意思是，存在一条直线把正负样本分开。当然有多条不同的直线，可以把正样本和负样本完全分开。



当画出这两条额外的蓝线，我们看到黑色的决策界和训练样本之间有更大的最短距离。然而粉线和蓝线离训练样本就非常近，在分离样本的时候就会比黑线表现差。因此，这个距离叫做支持向量机的间距，而这是支持向量机具有鲁棒性的原因，因为它努力用一个最大间距来分离样本。因此支持向量机有时被称为大间距分类器。

我们将这个大间距分类器中的正则化因子常数 C 设置的非常大，我记得我将其设置为了 100000，因此对这样的一个数据集，也许我们将选择这样的决策界，从而最大间距地分离开正样本和负样本。那么在让代价函数最小化的过程中，我们希望找出在 $y=1$ 和 $y=0$ 两种情况下都使得代价函数中左边的这一项尽量为零的参数。如果我们找到了这样的参数，则我们的最小化问题便转变成：

$$min\frac{1}{2}\sum_{j=1}^{n}\theta_j^2$$

$$\theta^T x^{(i)} >= 1, if \quad y^{(i)} = 1$$

$$\theta^T x^{(i)} <= -1, if \quad y^{(i)} = 0$$

事实上，支持向量机现在要比这个大间距分类器所体现得更成熟，尤其是当你使用大间距分类器的时候，你的学习算法会受异常点 (outlier) 的影响。比如我们加入一个额外的正样本。

在这里，如果我们加了这个样本，为了将样本用最大间距分开，也许最终会得到一条类似这样的决策界，对么？就是这条斜着的线，仅仅基于一个异常值，仅仅基于一个样本，就将我们的决策界从这条竖线变到这条斜线，这实在是不明智的。

而如果正则化参数 C，设置的非常大，这事实上正是支持向量机将会做的。它将决策界，从竖线变到了斜线；但是如果 C 设置的小一点，如果我们将 C 设置的不要太大，

则你最终会得到这条竖线，当然数据如果不是线性可分的，如果我们在这里有一些正样本或者我们在这里有一些负样本，则支持向量机也会将它们恰当分开。

因此，大间距分类器的描述，仅仅是从直观上给出了正则化参数 C 非常大的情形，同时，要提醒你 C 的作用类似于 $\frac{1}{\lambda}$，λ 是我们之前使用的正则化参数。

这只是 C 非常大的情形，或者等价地 λ 非常小的情形。你最终会得到类似斜线这样的决策界，但是实际上应用支持向量机的时候，当 C 不是非常非常大的时候，它可以忽略掉一些异常点的影响，得到更好的决策界。甚至当你的数据不是线性可分的时候，支持向量机也可以给出好的结果。

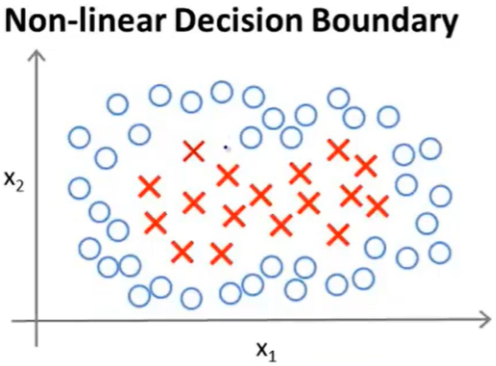
回顾 $C=\frac{1}{\lambda}$，因此：

C 较大时，相当于 λ 较小，可能会导致过拟合，高方差。

C 较小时，相当于 λ 较大，可能会导致欠拟合，高偏差。

1.3　核函数 1

我们之前讨论过可以使用高级数的多项式模型来解决无法用直线进行分隔的分类问题：



Predict $y=1$ if

$$\theta_0+\theta_1x_1+\theta_2x_2+\theta_3x_1x_2+\theta_4x_1^2+\theta_5x_2^2+\cdots\geq 0$$

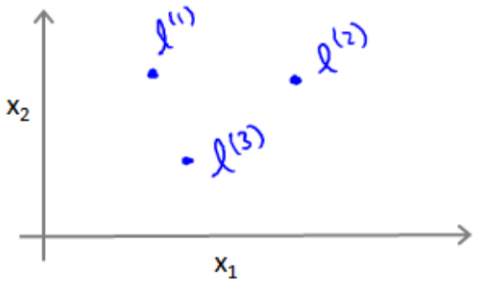
为了获得上图所示的判定边界，我们的模型可能是 $\theta_0+\theta_1x_1+\theta_2x_2+\theta_3x_1x_2+\theta_4x_1^2+\theta_5x_2^2+\cdots$ 的形式。

我们可以用一系列的新的特征 f 来替换模型中的每一项。例如令：

$f_1=x_1, f_2=x_2, f_3=x_1x_2, f_4=x_1^2, f_5=x_2^2$

得到 $h_\theta(x)=\theta_1f_1+\theta_2f_2+\theta_3f_3+\cdots+\theta_nf_n$。然而，除了对原有的特征进行组合以外，有没有更好的方法来构造 f_1, f_2, f_3? 我们可以利用核函数来计算出新的特征。

给定一个训练样本 x，我们利用 x 的各个特征与我们预先选定的地标 (landmarks)$l^{(1)}, l^{(2)}, l^{(3)}$ 的近似程度来选取新的特征 f_1, f_2, f_3。



例如：$f_1 = similarity(x, l^{(1)}) = e^{-\frac{||x-l^{(1)}||^2}{2\delta^2}}$

其中：$||x - l^{(1)}||^2 = \sum_{j=1}^{n}(x_j - l_j^{(1)})^2$，为实例中 x 中所有特征与地标 $l^{(1)}$ 之间的距离的和。

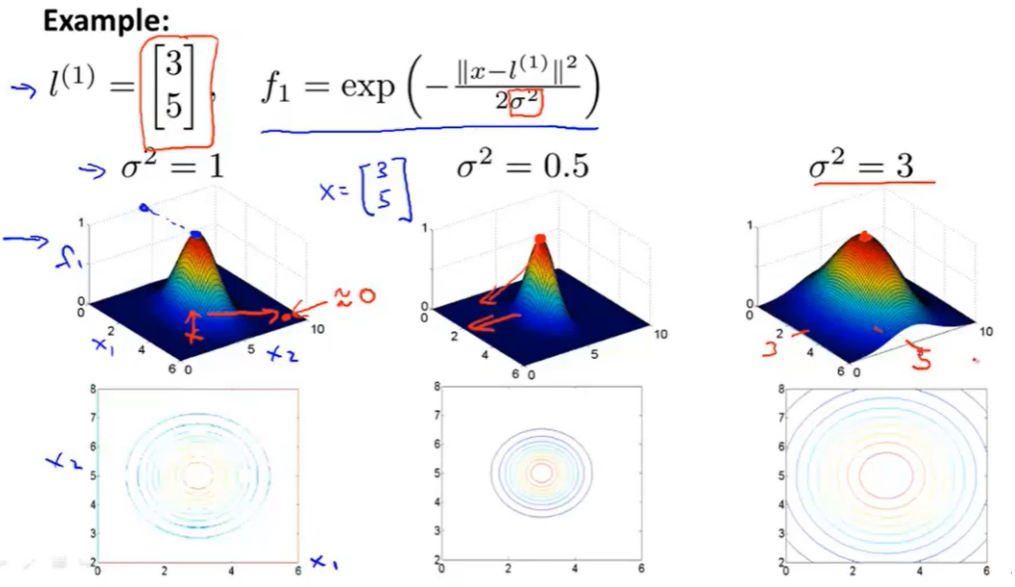
上例中的 $similarity(x, l^{(1)})$ 就是核函数，具体而言，这里是一个高斯核函数 (Gaussian Kernel)。

注：这个函数与正态分布没什么实际上的关系，只是看上去像而已。

这些地标的作用是什么？

如果一个训练样本 x 与地标 $l^{(1)}$ 之间的距离近似于 0，则新特征 f 近似于 $e^{-0} = 1$，如果训练样本 x 与地标 $l^{(1)}$ 之间距离较远，则 f 近似于 $e^{-big} = 0$。

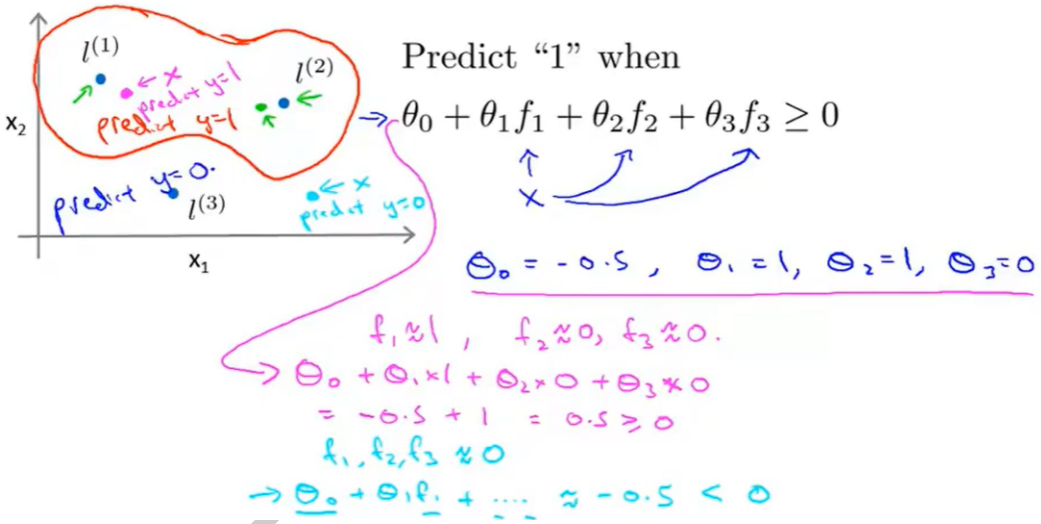
假设我们的训练样本含有两个特征 $[x_1, x_2]$，给定地标 $l^{(1)}$ 与不同的 δ 值，见下图：



图中水平面的坐标为 x_1, x_2 而垂直坐标轴代表 f。可以看出，只有当 x 与 $l^{(1)}$ 重合时 f 才具有最大值。随着 x 的改变 f 值改变的速率受到 δ^2 的影响。

在下图中，当样本处与 $l^{(1)}$ 下方的点位置处，因为其离 $l^{(1)}$ 更近，但是离 $l^{(2)}, l^{(3)}$ 较远，因此 f_1 接近 1，而 f_2, f_3 接近 0。因此 $h_\theta(x) = \theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 > 0$，因此预

测 $y=1$。同理可以求出，对于离 $l^{(2)}$ 较近的绿色点，也预测 $y=1$，但是对于最下面的点，因为其离三个地标都较远，预测 $y=0$。



这样，图中的封闭曲线所表示的范围，便是我们依据一个单一的训练样本和我们选取的地标所得出的判定边界，在预测时，我们采用的特征不是训练样本本身的特征，而是通过核函数计算出的新特征 f_1, f_2, f_3。

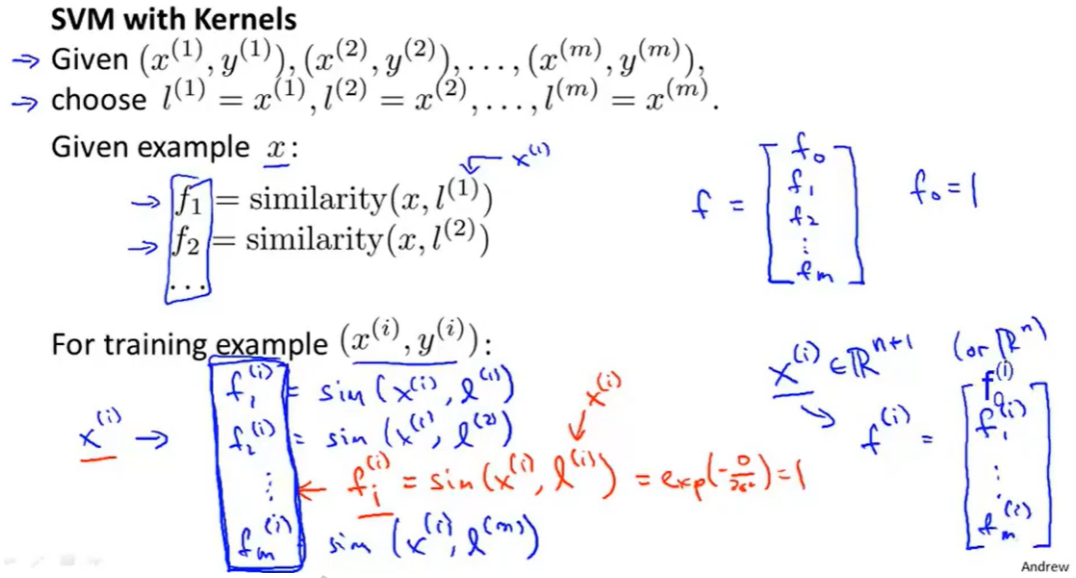
1.4　核函数 2

如何选择地标？

我们通常是根据训练集的数量选择地标的数量，即如果训练集中有 m 个样本，则我们选取 m 个地标，并且令:

$l^{(1)}=x^{(1)}, l^{(2)}=x^{(2)}, l^{(3)}=x^{(3)}, \cdots, l^{(m)}=x^{(m)}$。这样做的好处在于：现在我们得到的新特征是建立在原有特征与训练集中所有其他特征之间距离的基础之上的，即：

$f_0^{(i)}=1, f_1^{(i)}=sim(x^{(i)}, l^{(1)}), f_2^{(i)}=sim(x^{(i)}, l^{(2)}), \cdots, f_i^{(i)}=sim(x^{(i)}, l^{(i)})=e^0=1, \cdots, f_m^{(i)}=sim(x^{(i)}, l^{(m)})$。



下面我们将核函数运用到支持向量机中，修改我们的支持向量机假设为：

给定 x，计算新特征 f，当 $\theta^T f >= 0$ 时，预测 $y = 1$，否则反之。

相应地修改代价函数为：$\sum_{j=1}^{n=m} \theta_j^2 = \theta^T \theta$

$minC \sum_{i=1}^{m} [y^{(i)} cost_1(\theta^T f^{(i)}) + (1 - y^{(i)}) cost_0(\theta^T f^{(i)})] + \frac{1}{2} \sum_{j=1}^{n=m} \theta_j^2 = \theta^T \theta$

在具体实施过程中，我们还需要对最后的正则化项进行些微调整，在计算 $\sum_{j=1}^{n=m} \theta_j^2 = \theta^T \theta$ 时，我们用 $\theta^T M \theta$ 代替 $\theta^T \theta$，其中 M 是根据我们选择的核函数而不同的一个矩阵。这样做的原因是为了简化计算。

上面使用 M 来简化计算的方法不适用与逻辑回归，因此计算将非常耗费时间。

如果我们使用高斯核函数，那么在使用之前进行特征缩放是非常必要的。

另外，支持向量机也可以不使用核函数，不使用核函数又称为线性核函数 (linear kernel)，当我们不采用非常复杂的函数，或者我们的训练集特征非常多而样本非常少的时候，可以采用这种不带核函数的支持向量机。

下面是支持向量机的两个参数 C 和 δ 的影响：

$C = \frac{1}{\lambda}$

C 较大时，相当于 λ 较小，可能会导致过拟合，高方差；

C 较小时，相当于 λ 较大，可能会导致低拟合，高偏差；

δ 较大时，可能会导致低方差，高偏差；

δ 较小时，可能会导致低偏差，高方差。

1.5　使用支持向量机

假设我们利用之前介绍的一对多方法来解决一个多类分类问题。如果一共有 k 个类，则我们需要 k 个模型，以及 k 个参数向量 θ。我们同样也可以训练 k 个支持向量机来解决多类分类问题。

尽管我们不去写我们自己的 SVM 的优化软件，但是我们也需要做几件事：

1、是提出参数 C 的选择。我们在之前的笔记中讨论过误差/方差在这方面的性质。

2、我们也需要选择内核参数或我们想要使用的相似函数，其中一个选择是：我们选择不需要任何内核参数，没有内核参数的理念，也叫线性核函数。因此，如果有人说他使用了线性核的 SVM（支持向量机），这就意味这他使用了不带有核函数的 SVM（支持向量机）。

从逻辑回归模型，我们得到了支持向量机模型，在两者之间，我们应该如何选择呢？

下面是一些普遍使用的准则：

n 为特征数，m 为训练样本数。

(1) 如果相较于 m 而言，n 要大许多，即训练集数据量不够支持我们训练一个复杂的非线性模型，我们选用逻辑回归模型或者不带核函数的支持向量机。

(2) 如果 n 较小，而且 m 大小中等，例如 n 在 1-1000 之间，而 m 在 10-10000 之间，使用高斯核函数的支持向量机。

(3) 如果 n 较小，而 m 较大，例如 n 在 1-1000 之间，而 m 大于 50000，则使用支持向量机会非常慢，解决方案是创造、增加更多的特征，然后使用逻辑回归或不带核函数的支持向量机。

值得一提的是，神经网络在以上三种情况下都可能会有较好的表现，但是训练神经网络可能非常慢，选择支持向量机的原因主要在于它的代价函数是凸函数，不存在局部最小值。

逻辑回归和不带核函数的支持向量机它们都是非常相似的算法，不管是逻辑回归还是不带核函数的 SVM，通常都会做相似的事情，并给出相似的结果。但是根据我们实现的情况，其中一个可能会比另一个更加有效。但是在其中一个算法应用的地方，逻辑回归或不带核函数的 SVM 另一个也很有可能很有效。但是随着 SVM 的复杂度增加，当你使用不同的内核函数来学习复杂的非线性函数时，这个体系，你知道的，当你有多达 1 万（10,000）的样本时，也可能是 5 万（50,000），你的特征变量的数量这是相当大的。那是一个非常常见的体系，也许在这个体系里，不带核函数的支持向量机就会表现得相当突出。你可以做比这困难得多需要逻辑回归的事情。

神经网络使用于什么时候呢？

对于所有的这些问题，对于所有的这些不同体系一个设计得很好的神经网络也很有可能会非常有效。有一个缺点是，或者说是有时可能不会使用神经网络的原因是：对于许多这样的问题，神经网络训练起来可能会特别慢。但是如果我们有一个非常好的 SVM 实现包，它可能会运行得比较快比神经网络快很多，尽管我们在此之前没有展示，但是事实证明，SVM 具有的优化问题，是一种凸优化问题。因此，好的 SVM 优化软件包总是会找到全局最小值，或者接近它的值。对于 SVM 我们不需要担心局部最优解的问题。在实际应用中，局部最优不是神经网络所需要解决的一个重大问题，所以这是我们在使用 SVM 的时候不需要太去担心的一个问题。

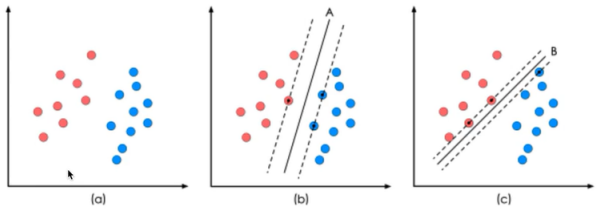
第二章　支持向量机的算法实现

需要说明的是，本次的实现使用了 $sklearn$ 库，$sklearn$ 库是一个机器学习算法包。它的集成度比较高，也是常用的 $Python$ 库之一。

知识点回顾：

首先是线性可分的概念，如下图所示：

线性可分：



然后是损失函数的定义如下图所示：

损失函数：

$$J(\theta)=\frac{1}{m}[\sum_{i=1}^{m}y^{(i)}(-logh_\theta(x^{(i)}))+(1-y^{(i)})(-(log(1-h_\theta(x^{(i)}))))]+\frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

$$J(\theta)=C\sum_{i=1}^{m}[y^{(i)}Cost_1(\theta^Tx^{(i)})+(1-y^{(i)})Cost_0(\theta^Tx^{(i)})]+\frac{1}{2}\sum_{j=1}^{n}\theta_j^2$$

C：误差项惩罚系数
越大，容错率越低，越易过拟合，低偏差，高方差

2.1　使用 SVM 对线性可分样本进行分类

我们将从一个二维示例数据集开始，该数据集可以用线性边界分隔。

在这个数据集中，正例（用 +）和负例（用 o 表示）的位置表明了由间隙表示的自然分离。

但是，请注意，在最左边处有一个异常值正值示例 +。作为本练习的一部分，我们还将看到此异常值如何影响 SVM 决策边界。

首先导入需要使用的包如下：

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.io import loadmat
from sklearn import svm
```

先看下数据的样子:

```
raw_data = loadmat('ex6data1.mat')
print('数据集的关键词:\n',raw_data.keys())
```

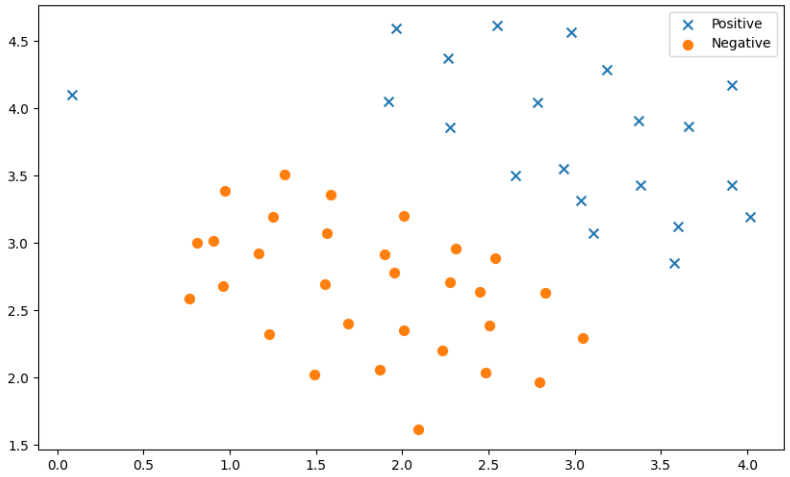
我们得到的结果如下：

```
数据集的关键词:
 dict_keys(['__header__', '__version__', '__globals__', 'X', 'y'])
```

我们对数据可视化的代码如下：

```
def plot_data():
    positive = data[data['y'].isin([1])]
    negative = data[data['y'].isin([0])]
    fig, ax = plt.subplots(figsize=(10,6))
    ax.scatter(positive['X1'], positive['X2'], s=50, marker='x', label='
                                          Positive')
    ax.scatter(negative['X1'], negative['X2'], s=50, marker='o', label='
                                          Negative')
    ax.legend()
    plt.show()
plot_data()
```

得到的结果如下：



这一部分让我们使用不同的参数 C 观察分类效果。C 就是 SVM 中对误分类样本的惩罚程度（正值）。C 越大对训练样本的分类就会越准确，但是泛化能力也会变差。

```
# 配置LinearSVC参数
svc = svm.LinearSVC(C=1,loss='hinge',max_iter=1000)#hinge loss是一种损失函
                                                    数
svc2= svm.LinearSVC(C=100,loss='hinge',max_iter=1000,random_state=40)
```

C=1 时：

```
clf = svc.fit(data[['X1','X2']],data['y'])
score = svc.score(data[['X1','X2']],data['y'])
print('C=1时:',score)
```

得到的结果为：

```
C=1时: 0.9803921568627451
```

同理，我们用相同的方法去计算 C=100 时，得到的结果为：

```
C=100时: 0.9607843137254902
```

只通过分类的准确率我们无法直观的看出分类效果，我们可视化一下决策边界，首先定义一下决策边界如下：

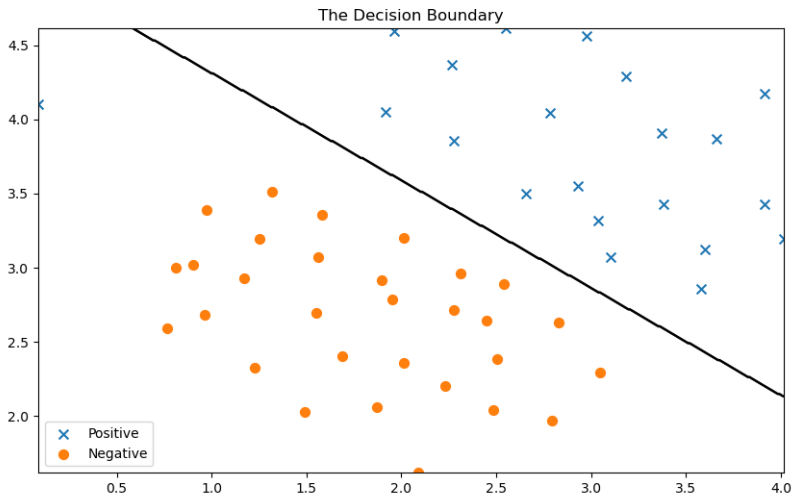
```
def plot_desicion_boundary(clf,x1min,x1max,x2min,x2max):
    u = np.linspace(x1min,x1max,500)
    v = np.linspace(x2min,x2max,500)
    # 转为网格（500*500）
    x,y = np.meshgrid(u,v)
```

```
    # 因为predict函数输入必须是二维数组
    # np.c_按行连接两个矩阵
    z = clf.predict(np.c_[x.flatten(),y.flatten()])
    z = z.reshape(x.shape) # 重新转为网格
    plt.contour(x,y,z,1,colors = 'black')
    plt.title('The Decision Boundary')
```

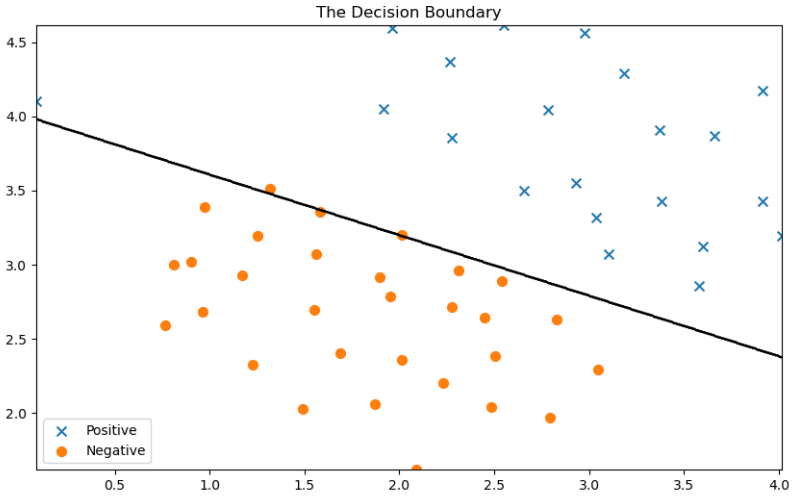
接下来我们可视化 C=1 时，代码实现如下：

```
X = raw_data['X']
y = raw_data['y'].flatten()
fig, ax = plt.subplots(figsize=(10,6))
positive = data[data['y'].isin([1])]
negative = data[data['y'].isin([0])]
ax.scatter(positive['X1'], positive['X2'], s=50, marker='x', label='
                                    Positive')
ax.scatter(negative['X1'], negative['X2'], s=50, marker='o', label='
                                    Negative')
plot_desicion_boundary(clf,np.min(X[:,0]),np.max(X[:,0]),np.min(X[:,1]),np.
                                    max(X[:,1]))
plt.legend(loc=3)
plt.show()
```

结果如下图所示：



同理可得，C=100 时的图像如下：



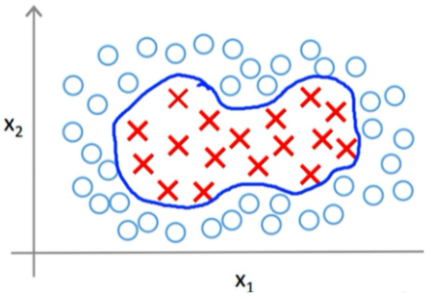
可以看到 C=100 时虽然没有了误分类样本，但这个决策边界却过拟合了，当样本点轻微波动时，可能就会分类错误。而 C=1 时虽然有一个误分类的样本，却能满足样本点波动的条件，也就是样本点到决策边界的间距较大。

2.2　高斯核函数 SVM

这一部分使用 SVM 进行非线性的分类。

核函数：

$h_\theta(x) = \theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 + \cdots$（用f代替x的参数）



将低维空间映射到高维空间

可以在低维空间计算出高维空间的点积结果

高斯核函数公式：

$$K_{gaussian}(x^{(i)}, x^{(j)}) = \exp\left(-\frac{\|x^{(i)} - x^{(j)}\|^2}{2\sigma^2}\right) = \exp\left(-\frac{\sum_{k=1}^{n}(x_k^{(i)} - x_k^{(j)})^2}{2\sigma^2}\right)$$

该函数的功能基本可以看作是测量两个样本点间的距离。参数 σ 表示宽度，这决定了随着示例之间的距离越来越远，相似性度量降低 (到 0) 的速度有多快。

```
def Gausskernal(xi,xj,sigma):
    return np.exp(np.sum(np.power(xi - xj,2)) / (- 2 * sigma ** 2))
xi = np.array([1, 2, 1])
xj = np.array([0, 4, -1])
sigma = 2
print(Gausskernal(xi, xj, sigma))
```

我们得到的最终结果为：

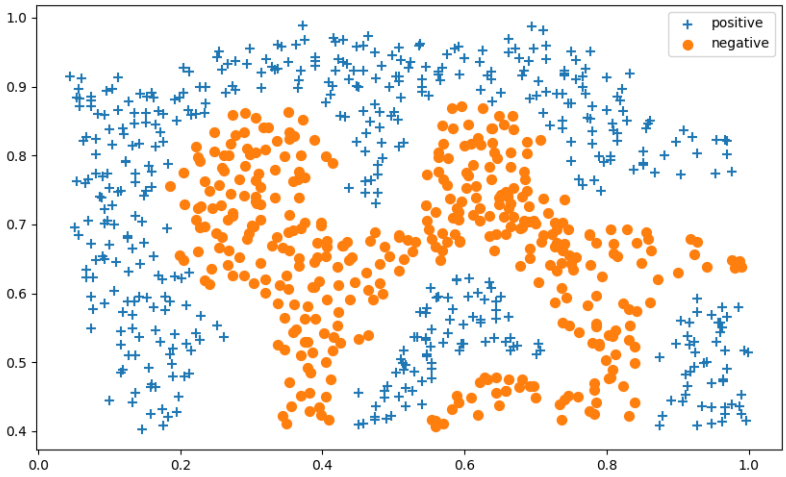
```
0.32465246735834974
```

该结果与练习中的预期值相符。接下来，我们将检查另一个数据集，这次用非线性决策边界。

2.3　非线性决策边界

老样子，先看数据：

```
raw_data = loadmat('ex6data2.mat')
data = pd.DataFrame(raw_data['X'], columns=['X1', 'X2'])
data['y'] = raw_data['y']
print('数据的特征:\n',raw_data.keys())
def plot_data():
    positive = data[data['y'].isin([1])]
    negative = data[data['y'].isin([0])]
    fig, ax = plt.subplots(figsize=(10, 6))
    ax.scatter(positive['X1'], positive['X2'], s=50, marker='+', label='
                                        positive')
    ax.scatter(negative['X1'], negative['X2'], s=50, marker='o', label='
                                        negative')
    ax.legend()
    plt.show()
plot_data()
```

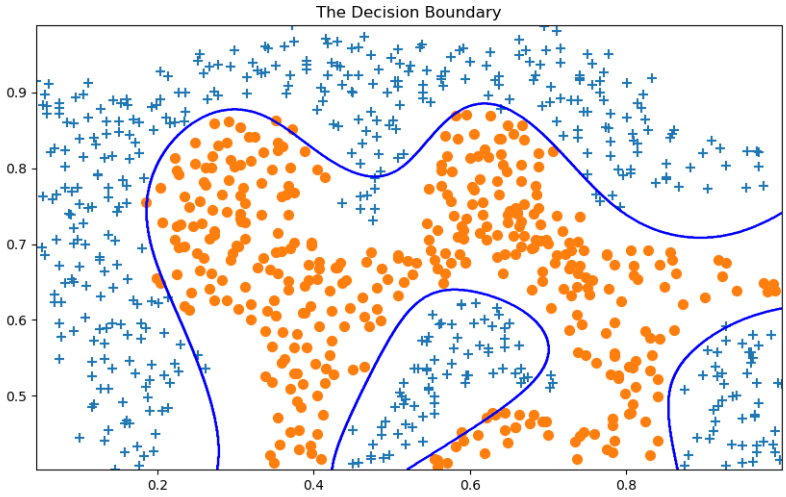


从图中，我们可以看到没有线性决策边界来分隔此数据集的正示例和负示例。然而，通过使用高斯核和支持向量机，将能够学习一个非线性的决策边界，它可以很好地执行数据集。

```
svc = svm.SVC(C=100, gamma=10, probability=True)
clf = svc.fit(data[['X1', 'X2']], data['y'])
score = svc.score(data[['X1', 'X2']], data['y'])
print(score)
```

得到的 *score* 的结果如下所示：

```
0.9698725376593279
```



上图显示了使用高斯核的 SVM 发现的决策边界。

决策边界能够正确地分离大多数正、负样本，并很好地跟踪数据集的轮廓。

2.4　最优超参数

这一部分使用交叉验证集来找到超参数 C 和 σ 的最优值。

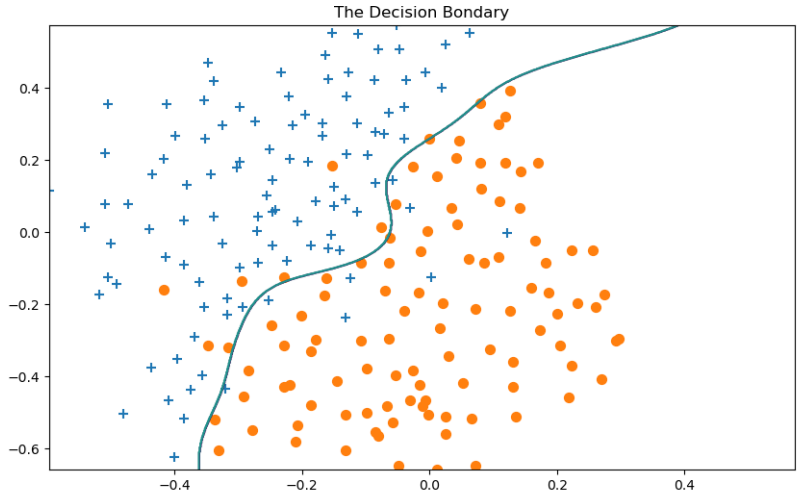
思路就是给定一个集合，使用集合里的数值作为超参数在训练集中训练模型，在交叉验证集中计算误差，使得误差最小的参数就是最优的。

```
c_values = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30]
gamma_values = c_values
best_score = 0
best_params = {'C': None, 'gamma': None}
for c in c_values:
    for select_gamma in gamma_values:
        svc = svm.SVC(C=c, gamma=select_gamma, probability=True)
        # 用训练集训练
        clf3 = svc.fit(x, y)
        # 用验证集优选
        score = svc.score(Xval, yval)
        if score > best_score:
           best_score = score
           best_params['C'] = c
           best_params['gamma'] = select_gamma
print(best_score, best_params)
c = best_params['C']
gamma = best_params['gamma']
svc = svm.SVC(C=c, gamma=gamma, probability=True)
clf = svc.fit(Xval, yval)
```

我们得出的分数和最佳参数为：

```
0.965 {'C': 3, 'gamma': 30}
```

用上面的参数绘制出决策函数图像如下所示：



可以看出，虽然有误分类的数据，但是整体效果较好。

2.5　垃圾邮件分类

现在，我们将进行新的练习。在这一部分中，我们的目标是使用 SVM 来构建垃圾邮件过滤器。

在练习文本中，有一个任务涉及一些文本预处理，以获得适合 SVM 处理的格式的数据。

```
spam_train=loadmat('spamTrain.mat')
spam_test=loadmat('spamTest.mat')
print('训练集的关键词:\n',spam_train.keys())
print('测试集的关键词:\n',spam_test.keys())
X=spam_train['X']#(4000,1899)
X_test=spam_test['Xtest']#(1000,1899)
y=spam_train['y']#(4000,1)
y_test=spam_test['ytest']#(1000,1)
```

其中 1899 个维对应于词汇表中的 1899 个单词. 它们的值为二进制，表示文档中是否存在单词。

```
svc=svm.SVC()
svc.fit(X, y)
print('Training accuracy = {0}%'.format(np.round(svc.score(X, y) * 100, 2))
                                        )
```

```
print('Test accuracy = {0}%'.format(np.round(svc.score(X_test, y_test) *
                                        100, 2)))
```

最后得出我们的模型准确率为：

```
Training accuracy = 99.32%
Test accuracy = 98.7%
```

第三章　附录—源程序

3.1　—

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sb
from scipy.io import loadmat
from sklearn import svm
raw_data = loadmat('ex6data1.mat')
print('数据集的关键词:\n',raw_data.keys())
data = pd.DataFrame(raw_data['X'], columns=['X1', 'X2'])
data['y'] = raw_data['y']
def plot_data():
    positive = data[data['y'].isin([1])]
    negative = data[data['y'].isin([0])]
    fig, ax = plt.subplots(figsize=(10,6))
    ax.scatter(positive['X1'], positive['X2'], s=50, marker='x', label='
                                        Positive')
    ax.scatter(negative['X1'], negative['X2'], s=50, marker='o', label='
                                        Negative')
    ax.legend()
    plt.show()
plot_data()
# 配置LinearSVC参数
svc = svm.LinearSVC(C=1,loss='hinge',max_iter=1000)#hinge loss是一种损失函
                                        数
svc2= svm.LinearSVC(C=100,loss='hinge',max_iter=1000,random_state=40)
# 将配置好的模型应用于数据集
clf = svc.fit(data[['X1','X2']],data['y'])#clf = classifier的缩写
clf2= svc2.fit(data[['X1','X2']],data['y'])
```

```
score = svc.score(data[['X1','X2']],data['y'])
score2= svc2.score(data[['X1','X2']],data['y'])
print('C=1时:',score)
print('C=100时:',score2)
#  画出决策边界
def plot_desicion_boundary(clf,x1min,x1max,x2min,x2max):
    u = np.linspace(x1min,x1max,500)
    v = np.linspace(x2min,x2max,500)
    # 转为网格（500*500）
    x,y = np.meshgrid(u,v)
    # 因为predict函数输入必须是二维数组
    # np.c_按行连接两个矩阵
    z = clf.predict(np.c_[x.flatten(),y.flatten()])
    z = z.reshape(x.shape) # 重新转为网格
    plt.contour(x,y,z,1,colors = 'black')
    plt.title('The Decision Boundary')
X = raw_data['X']
y = raw_data['y'].flatten()
fig, ax = plt.subplots(figsize=(10,6))
positive = data[data['y'].isin([1])]
negative = data[data['y'].isin([0])]
ax.scatter(positive['X1'], positive['X2'], s=50, marker='x', label='
                                        Positive')
ax.scatter(negative['X1'], negative['X2'], s=50, marker='o', label='
                                        Negative')
#plot_desicion_boundary(clf,np.min(X[:,0]),np.max(X[:,0]),np.min(X[:,1]),np
                                        .max(X[:,1]))
plot_desicion_boundary(clf2,np.min(X[:,0]),np.max(X[:,0]),np.min(X[:,1]),np
                                        .max(X[:,1]))
plt.legend(loc=3)
plt.show()
#现在我们将从线性SVM转移到能够使用内核进行非线性分类的SVM。
def Gausskernal(xi,xj,sigma):
    return np.exp(np.sum(np.power(xi - xj,2)) / (- 2 * sigma ** 2))
xi = np.array([1, 2, 1])
xj = np.array([0, 4, -1])
sigma = 2
print(Gausskernal(xi, xj, sigma))
# 0.32465246735834974
```

3.2　二

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sb
from scipy.io import loadmat
from sklearn import svm
def SVM_1():
    raw_data = loadmat('ex6data2.mat')
    data = pd.DataFrame(raw_data['X'], columns=['X1', 'X2'])
    data['y'] = raw_data['y']
    print('数据的特征:\n',raw_data.keys())
    def plot_data():
        positive = data[data['y'].isin([1])]
        negative = data[data['y'].isin([0])]
        fig, ax = plt.subplots(figsize=(10, 6))
        ax.scatter(positive['X1'], positive['X2'], s=50, marker='+', label=
                                                'positive')
        ax.scatter(negative['X1'], negative['X2'], s=50, marker='o', label=
                                                'negative')
        ax.legend()
        plt.show()
    plot_data()
    svc = svm.SVC(C=100, gamma=10, probability=True)
    clf = svc.fit(data[['X1', 'X2']], data['y'])
    score = svc.score(data[['X1', 'X2']], data['y'])
    print(score)

    def plot_desicion_boundary(clf, x1min, x1max, x2min, x2max):
        x1min = np.min(data['X1'])
        x1max = np.max(data['X1'])
        x2min = np.min(data['X2'])
        x2max = np.max(data['X2'])
        u = np.linspace(x1min, x1max, 1000)
        v = np.linspace(x2min, x2max, 1000)
        # 转为网格（500*500）
        x, y = np.meshgrid(u, v)
        # 因为predict函数输入必须是二维数组
```

```
        # np.c_按行连接两个矩阵
        z = clf.predict(np.c_[x.flatten(), y.flatten()])
        z = z.reshape(x.shape)  # 重新转为网格
        plt.contour(x, y, z, 1, colors='b')
        plt.title('The Decision Boundary')

    X_test = raw_data['X']
    positive = data[data['y'].isin([1])]
    negative = data[data['y'].isin([0])]
    fig, ax = plt.subplots(figsize=(10, 6))
    ax.scatter(positive['X1'], positive['X2'], s=50, marker='+', label='
                                        positive')
    ax.scatter(negative['X1'], negative['X2'], s=50, marker='o', label='
                                        negative')
    plot_desicion_boundary(clf, np.min(X_test[:, 0]), np.max(X_test[:, 0]),
                                         np.min(X_test[:, 1]), np.max(
                                        X_test[:, 1]))
    plt.show()
    return None

def SVM_2():
    #在这部分练习中，您将获得有关如何使用带高斯核的支持向量机的更多实用技
                                        能。
    raw_data=loadmat('ex6data3.mat')
    data=pd.DataFrame(raw_data['X'], columns=['X1', 'X2'])
    data['y']=raw_data['y']
    print(raw_data.keys())
    def plot_data():
        positive=data[data['y'].isin([1])]
        negative=data[data['y'].isin([0])]
        fig,ax=plt.subplots(figsize=(10,6))
        ax.scatter(positive['X1'],positive['X2'],s=50,marker='+',label='
                                            positive')
        ax.scatter(negative['X1'],negative['X2'],s=50,marker='o',label='
                                            negative')
        ax.legend(loc=2)
        plt.show()
    x,y=raw_data['X'],raw_data['y']
    Xval, yval = raw_data['Xval'], raw_data['yval']
```

```
#找出最佳的参数
c_values = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30]
gamma_values = c_values
best_score = 0
best_params = {'C': None, 'gamma': None}
for c in c_values:
    for select_gamma in gamma_values:
        svc = svm.SVC(C=c, gamma=select_gamma, probability=True)
        # 用训练集训练
        clf3 = svc.fit(x, y)
        # 用验证集优选
        score = svc.score(Xval, yval)
        if score > best_score:
            best_score = score
            best_params['C'] = c
            best_params['gamma'] = select_gamma
print(best_score, best_params)
c = best_params['C']
gamma = best_params['gamma']
svc = svm.SVC(C=c, gamma=gamma, probability=True)
clf = svc.fit(Xval, yval)
def plot_decision_boundary(clf,x1min,x1max,x2min,x2max):
    x1min=np.min(data['X1'])
    x1max=np.max(data['X2'])
    x2min=np.min(data['X2'])
    x2max=np.max(data['X2'])
    u=np.linspace(x1min,x1max,1000)
    v=np.linspace(x2min,x2max,1000)
    x,y=np.meshgrid(u,v)
    z=clf.predict(np.c_[x.flatten(),y.flatten()])
    z=z.reshape(x.shape)
    plt.contour(x,y,z,1,c='black') #作用：绘制轮廓线，类于等高线
    plt.title('The Decision Bondary')
    plt.show()
positive = data[data['y'].isin([1])]
negative = data[data['y'].isin([0])]
fig, ax = plt.subplots(figsize=(10, 6))
ax.scatter(positive['X1'], positive['X2'], s=50, marker='+', label='
                                        positive')
```

```
    ax.scatter(negative['X1'], negative['X2'], s=50, marker='o', label='
                                        negative')
    plot_decision_boundary(clf, np.min(Xval[:, 0]), np.max(Xval[:, 1]), np.
                                        min(Xval[:, 0]), np.max(Xval[:, 1
                                        ]))
    plt.show()
    return None
def Spam_Classification():
    """
    在这一部分中，我们的目标是使用SVM来构建垃圾邮件过滤器。
    在练习文本中，有一个任务涉及一些文本预处理，以获得适合SVM处理的格式的数
                                        据。
    :return:
    """
    spam_train=loadmat('spamTrain.mat')
    spam_test=loadmat('spamTest.mat')
    print('训练集的关键词:\n',spam_train.keys())
    print('测试集的关键词:\n',spam_test.keys())
    X=spam_train['X']#(4000,1899)
    X_test=spam_test['Xtest']#(1000,1899)
    y=spam_train['y']#(4000,1)
    y_test=spam_test['ytest']#(1000,1)
    #其中1,899个维对应于词汇表中的1,899个单词.它们的值为二进制，表示文档中
                                        是否存在单词。
    svc=svm.SVC()
    svc.fit(X, y)
    print('Training accuracy = {0}%'.format(np.round(svc.score(X, y) * 100,
                                         2)))
    print('Test accuracy = {0}%'.format(np.round(svc.score(X_test, y_test)
                                        * 100, 2)))
    return None
if __name__=='__main__':
    Spam_Classification()
```