菜菜的scikit-learn课堂07

sklearn中的支持向量机SVM（一）

小伙伴们晚上好~o(￣▽￣)ブ

我是菜菜，这里是我的sklearn课堂第七期，今晚的直播内容是支持向量机（上），下周还有下篇哦~

我的开发环境是**Jupyter lab**，所用的库和版本大家参考：

Python 3.7.1（你的版本至少要3.4以上

Scikit-learn 0.20.1 （你的版本至少要0.20

Numpy 1.15.4, **Pandas** 0.23.4, **Matplotlib** 3.0.2, **SciPy** 1.1.0

请扫码进群领取课件和代码源文件，**扫描二维码后回复"K"就可以进群哦~**

菜菜的scikit-learn课堂07

sklearn中的支持向量机SVM（一）

本周内容

下周内容

sklearn中的支持向量机SVM（二）

本周内容

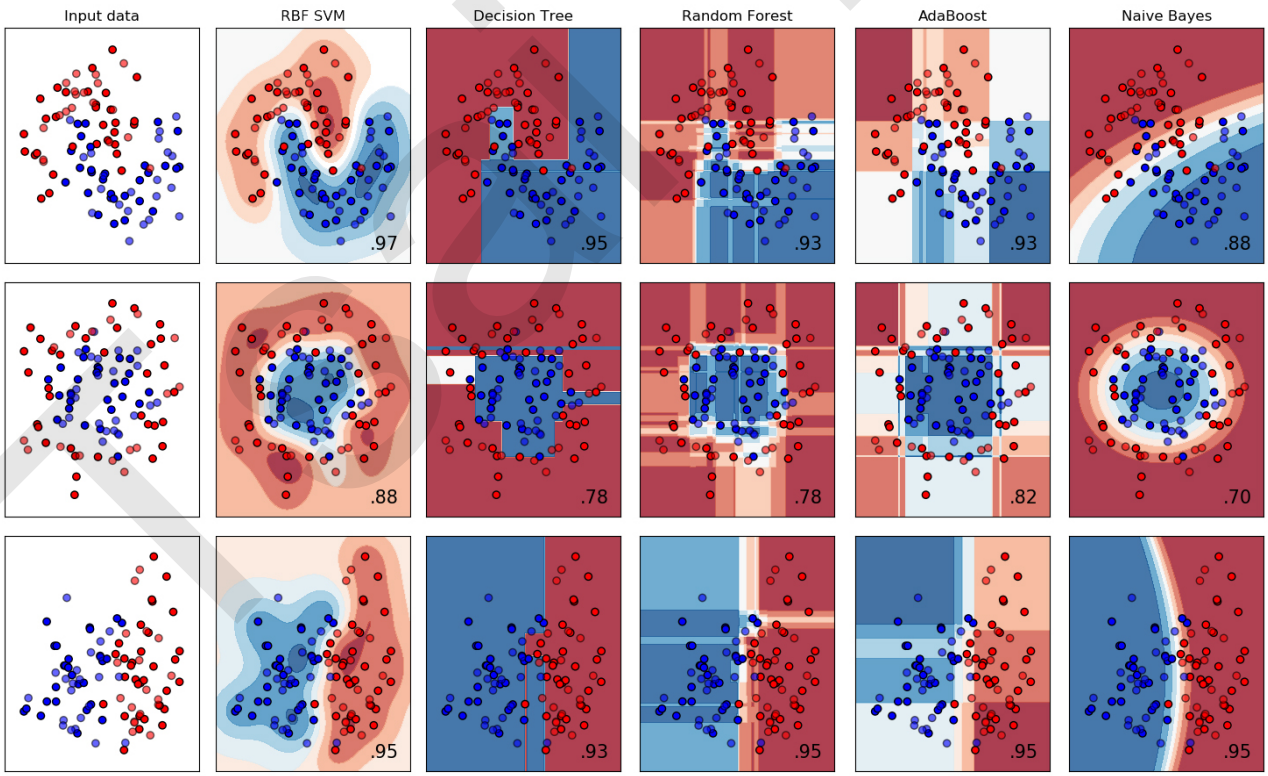
1 概述

支持向量机（SVM，也称为支持向量网络），是机器学习中获得关注最多的算法没有之一。它源于统计学习理论，是我们除了集成算法之外，接触的第一个强学习器。它有多强呢？

从算法的功能来看，SVM几乎囊括了我们前六周讲解的所有算法的功能：

	功能
有监督学习	线性二分类与多分类（Linear Support Vector Classification） 非线性二分类与多分类（Support Vector Classification, SVC） 普通连续型变量的回归（Support Vector Regression） 概率型连续变量的回归（Bayesian SVM）
无监督学习	支持向量聚类（Support Vector Clustering，SVC） 异常值检测（One-class SVM）
半监督学习	转导支持向量机（Transductive Support Vector Machines，TSVM）

从分类效力来讲，SVM在无论线性还是非线性分类中，都是明星般的存在：



从实际应用来看，SVM在各种实际问题中都表现非常优秀。它在**手写识别数字**和**人脸识别**中应用广泛，在**文本和超文本的分类**中举足轻重，因为SVM可以大量减少标准归纳（standard inductive）和转换设置（transductive settings）中对标记训练实例的需求。同时，SVM也被用来执行**图像的分类，并用于图像分割系统。**实验结果表明，在仅仅三到四轮相关反馈之后，SVM就能实现比传统的查询细化方案（query refinement schemes）高出一

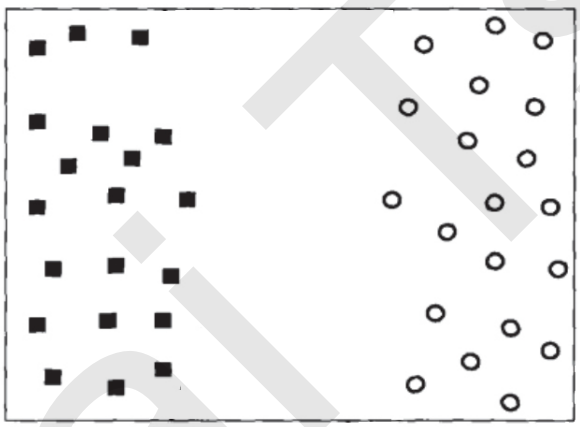
大截的搜索精度。除此之外，生物学和许多其他科学都是SVM的青睐者，SVM现在已经广泛被用于**蛋白质分类**，现在化合物分类的业界平均水平可以达到90%以上的准确率。在生物科学的尖端研究中，人们还使用支持向量机来**识别用于模型预测的各种特征**，以找出各种基因表现结果的影响因素。

从学术的角度来看，SVM是**最接近深度学习的机器学习算法。**线性SVM可以看成是神经网络的单个神经元（虽然损失函数与神经网络不同），非线性的SVM则与两层的神经网络相当，非线性的SVM中如果添加多个核函数，则可以模仿多层的神经网络。而从数学的角度来看，SVM的数学原理是公认的对初学者来说难于上青天的水平，对于没有数学基础和数学逻辑熏陶的人来说，探究SVM的数学原理本身宛如在知识的荒原上跋涉。

当然了，没有算法是完美的，比SVM强大的算法在集成学习和深度学习中还有很多很多。但不可否认，它是我们目前为止接触到的最强大的算法。接下来的两周，我们将一起来探索SVM的神秘世界。

1.1 支持向量机分类器是如何工作的

支持向量机所作的事情其实非常容易理解。先来看看下面这一组数据的分布，这是一组两种标签的数据，两种标签分别由圆和方块代表。**支持向量机的分类方法，是在这组分布中找出一个超平面作为决策边界，使模型在数据上的分类误差尽量接近于小，尤其是在未知数据集上的分类误差（泛化误差）尽量小。**



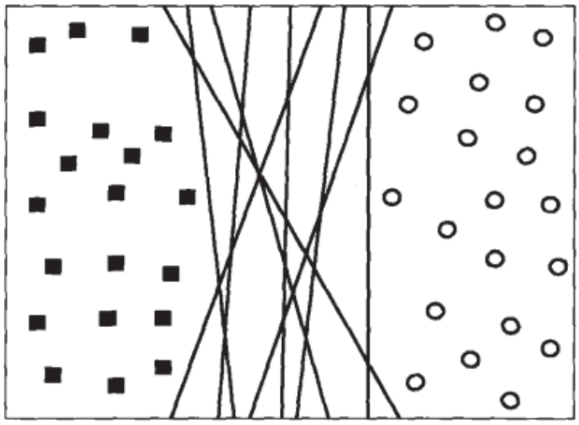
关键概念：超平面

在几何中，超平面是一个空间的子空间，它是维度比所在空间小一维的空间。 如果数据空间本身是三维的，则其超平面是二维平面，而如果数据空间本身是二维的，则其超平面是一维的直线。

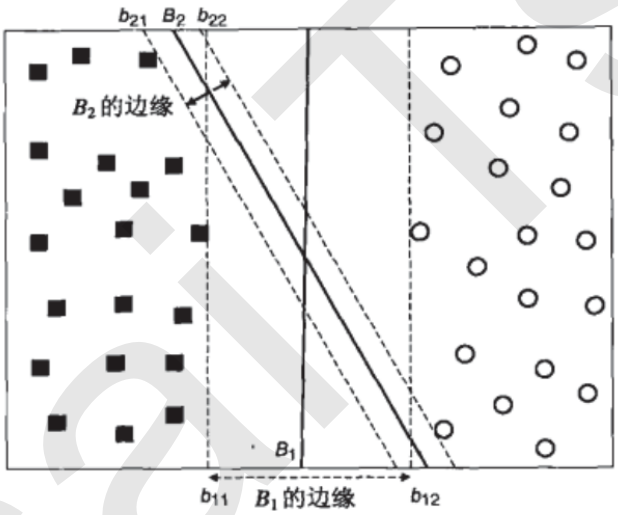
在二分类问题中，如果一个超平面能够将数据划分为两个集合，其中每个集合中包含单独的一个类别，我们就说这个超平面是数据的“决策边界”。

决策边界一侧的所有点在分类为属于一个类，而另一侧的所有点分类属于另一个类。如果我们能够找出决策边界，分类问题就可以变成探讨每个样本对于决策边界而言的相对位置。比如上面的数据分布，我们很容易就可以在方块和圆的中间画出一条线，并让所有落在直线左边的样本被分类为方块，在直线右边的样本被分类为圆。如果把数据当作我们的训练集，只要直线的一边只有一种类型的数据，就没有分类错误，我们的训练误差就会为0。

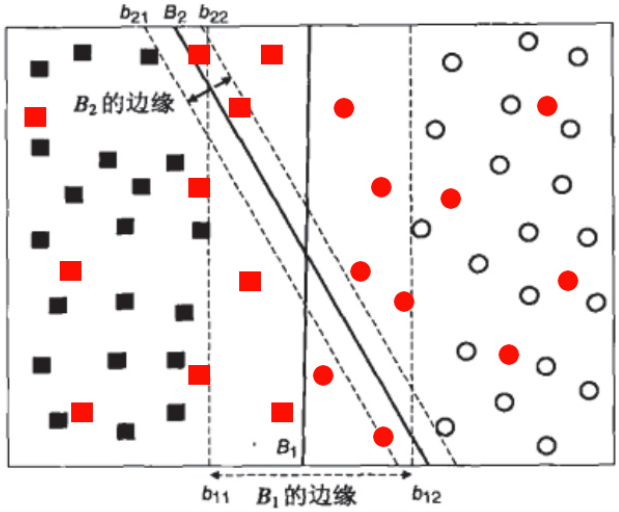
但是，对于一个数据集来说，让训练误差为0的决策边界可以有无数条。



但在此基础上，我们无法保证这条决策边界在未知数据集（测试集）上的表现也会优秀。对于现有的数据集来说，我们有B_1和B_2两条可能的决策边界。我们可以把决策边界B_1向两边平移，直到碰到离这条决策边界最近的方块和圆圈后停下，形成两个新的超平面，分别是b_{11}和b_{12}，并且我们将原始的决策边界移动到b_{11}和b_{12}的中间，确保B_1到b_{11}和b_{12}的距离相等。**在b_{11}和b_{12}中间的距离，叫做B_1这条决策边界的边际(margin)**，通常记作d。对B_2也执行同样的操作，然后我们来对比一下两个决策边界。现在两条决策边界右边的数据都被判断为圆，左边的数据都被判断为方块，两条决策边界在现在的数据集上的训练误差都是0，没有一个样本被分错。



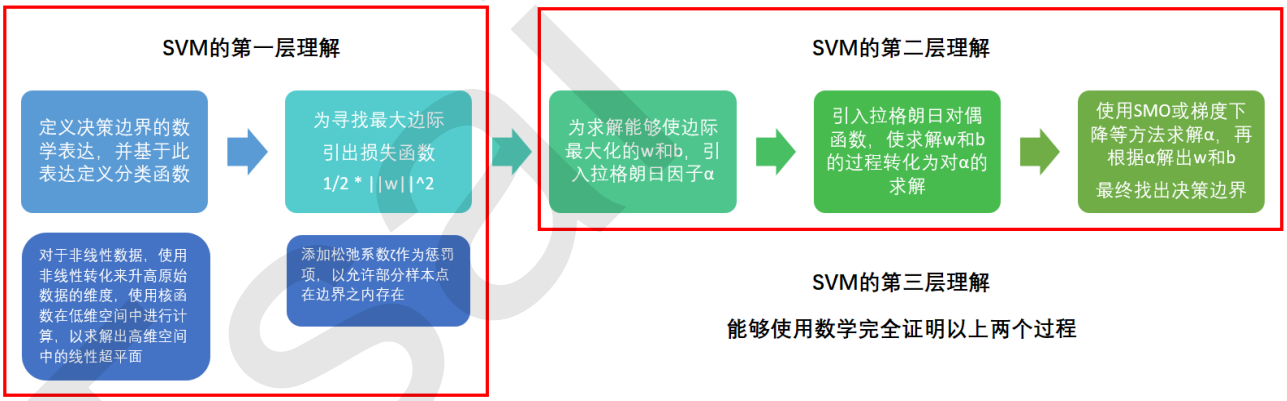
我们引入和原本的数据集相同分布的测试样本（红色所示），平面中的样本变多了，此时我们可以发现，对于B_1而言，依然没有一个样本被分错，这条决策边界上的泛化误差也是0。但是对于B_2而言，却有三个方块被误人类成了圆，二有两个圆被误分类成了方块，这条决策边界上的泛化误差就远远大于B_1了。这个例子表现出，**拥有更大边际的决策边界在分类中的泛化误差更小**，这一点可以由结构风险最小化定律来证明（SRM）。如果边际很小，则任何轻微扰动都会对决策边界的分类产生很大的影响。**边际很小的情况，是一种模型在训练集上表现很好，却在测试集上表现糟糕的情况，所以会"过拟合"。**所以我们在找寻决策边界的时候，希望边际越大越好。



支持向量机，就是通过找出边际最大的决策边界，来对数据进行分类的分类器。也因此，支持向量分类器又叫做最大边际分类器。这个过程在二维平面中看起来十分简单，但将上述过程使用数学表达出来，就不是一件简单的事情了。

1.2 支持向量机原理的三层理解

目标是"找出边际最大的决策边界"，听起来是一个十分熟悉的表达，这是一个最优化问题，而最优化问题往往和损失函数联系在一起。和逻辑回归中的过程一样，SVM也是通过最小化损失函数来求解一个用于后续模型使用的重要信息：决策边界。



1.2 sklearn中的支持向量机

类	含义	输入
svm.LinearSVC	线性支持向量分类	[penalty, loss, dual, tol, C, ⋯])
svm.LinearSVR	线性支持向量回归	[epsilon, tol, C, loss, ⋯])
svm.SVC	非线性多维支持向量分类	[C, kernel, degree, gamma, coef0, ⋯])
svm.SVR	非线性多维支持向量回归	[kernel, degree, gamma, coef0, tol, ⋯])
svm.NuSVC	Nu支持向量分类	[nu, kernel, degree, gamma, ⋯])
svm.NuSVR	Nu支持向量回归	[nu, C, kernel, degree, gamma, ⋯])
svm.OneClassSVM	无监督异常值检测	[kernel, degree, gamma, ⋯])
svm.l1_min_c	返回参数C的最低边界，使得对于C in （l1_min_C，infinity），模型保证不为空	X, y[, loss, fit_intercept, ⋯])
直接使用libsvm的函数		
svm.libsvm.cross_validation	SVM专用的交叉验证	
svm.libsvm.decision_function	SVM专用的预测边际函数（libsvm名称为predict_values）	
svm.libsvm.fit	使用libsvm训练模型	
svm.libsvm.predict	给定模型预测X的目标值	
svm.libsvm.predict_proba	预测概率	

注意，除了特别表明是线性的两个类LinearSVC和LinearSVR之外，其他的所有类都是同时支持线性和非线性的。NuSVC和NuSVC可以手动调节支持向量的数目，其他参数都与最常用的SVC和SVR一致。注意OneClassSVM是无监督的类。

除了本身所带的类之外，sklearn还提供了直接调用libsvm库的几个函数。Libsvm是台湾大学林智仁(Lin Chih-Jen)教授等人开发设计的一个简单、易于使用和快速有效的英文的SVM库，它提供了大量SVM的底层计算和参数选择，也是sklearn的众多类背后所调用的库。目前，LIBSVM拥有C、Java、Matlab、Python、R等数十种语言版本，每种语言版本都可以在libsvm的官网上进行下载：

https://www.csie.ntu.edu.tw/~cjlin/libsvm/

2 sklearn.svm.SVC

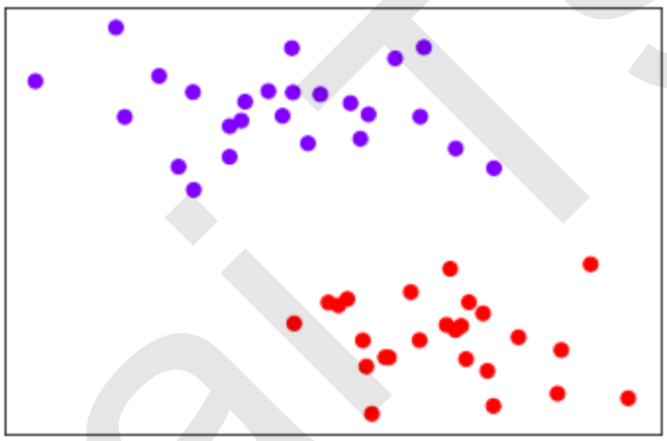
class `sklearn.svm.SVC` *(C=1.0, kernel='rbf', degree=3, gamma='auto_deprecated', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', random_state=None)*

2.1 线性SVM用于分类

2.1.1 线性SVM的损失函数

要理解SVM的损失函数，我们先来定义决策边界。假设现在数据中总计有N个训练样本，每个训练样本i可以被表示为$(\boldsymbol{x_i}, y_i)(i = 1, 2, ...N)$，其中$\boldsymbol{x_i}$是$(x_{1i}, x_{2i} .... x_{ni})^T$这样的一个特征向量，每个样本总共含有$n$个特征。二分类标签$y_i$的取值是{-1, 1}。

如果n等于2，则有$i = (x_{1i}, x_{2i}, y_i)^T$，分别由我们的特征向量和标签组成。此时我们可以在二维平面上，以x_2为横坐标，x_1为纵坐标，y为颜色，来可视化我们所有的N个样本：



我们让所有紫色点的标签为1，红色点的标签为-1。我们要在这个数据集上寻找一个决策边界，在二维平面上，决策边界（超平面）就是一条直线。二维平面上的任意一条线可以被表示为：

$$x_1 = ax_2 + b$$

我们将此表达式变换一下：

$$\begin{aligned} 0 &= ax_2 - x_1 + b \\ 0 &= [a, -1] * \begin{bmatrix} x_2 \\ x_1 \end{bmatrix} + b \\ 0 &= \boldsymbol{w^T x} + b \end{aligned}$$

其中[a, -1]就是我们的参数向量$\boldsymbol{w}$，$\boldsymbol{x}$就是我们的特征向量，b是我们的截距。注意，这个表达式长得非常像我们线性回归的公式：

$$y(x) = \boldsymbol{\theta^T x} + \theta_0$$

线性回归中等号的一边是标签，回归过后会拟合出一个标签，而决策边界的表达式中却没有标签的存在，全部是由参数，特征和截距组成的一个式子，等号的一边是0。在一组数据下，给定固定的w和b，这个式子就可以是一条固定直线，在w和b不确定的状况下，这个表达式$w^T x + b = 0$就可以代表平面上的任意一条直线。如果在w和b固定时，给定一个唯一的x的取值，这个表达式就可以表示一个固定的点。**在SVM中，我们就使用这个表达式来表示我们的决策边界**。我们的目标是求解能够让边际最大化的决策边界，所以我们要求解参数向量w和截距b。

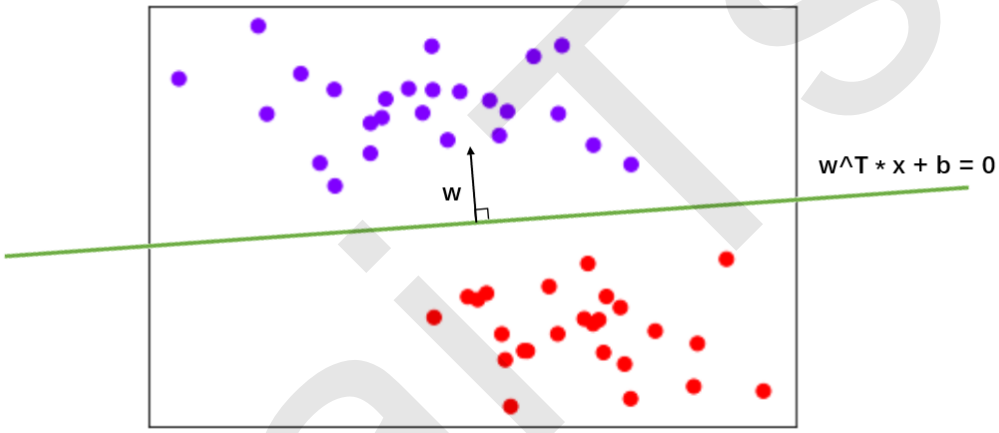
如果在决策边界上任意取两个点x_a，x_b，并带入决策边界的表达式，则有：

$$w^T x_a + b = 0$$
$$w^T x_b + b = 0$$

将两式相减，可以得到：

$$w^T * （x_a - x_b） = 0$$

一个列向量的转至乘以另一个列向量，可以获得两个向量的点积(dot product)，表示为$\langle w \cdot (x_a - x_b) \rangle$。两个向量的点击为0表示两个向量的方向式互相垂直的。x_a与x_b是一条直线上的两个点，相减后的得到的向量方向是由x_b指向x_a，所以$x_a - x_b$的方向是平行于他们所在的直线——我们的决策边界的。而w与$x_a - x_b$相互垂直，所以参数向量w**的方向必然是垂直于我们的决策边界。**



此时，我们有了我们的决策边界。任意一个紫色的点x_p就可以被表示为：

$$w \cdot x_p + b = p$$

由于紫色的点所代表的标签y是1，所以**我们规定，p>0**。同样的，对于任意一个红色的点x_r而言，我们可以将它表示为：

$$w \cdot x_r + b = r$$

由于红色点所表示的标签y是-1，所以**我们规定，r<0**。由此，如果我们有新的测试数据x_t，则的x_t标签就可以根据以下式子来判定：

$$y = \begin{cases} 1, & if\ w \cdot x_t + b > 0 \\ -1, & if\ w \cdot x_t + b < 0 \end{cases}$$

核心误区：p和r的符号

注意，在这里，p和r的符号是我们人为规定的。在一些博客或教材中，会认为p和r的符号是由原本的决策边界上下移动得到。这是一种误解。

如果k和k'是由原本的决策边界平移得到的话，紫色的点在决策边界上方，$\boldsymbol{w}\cdot\boldsymbol{x}+b=0$应该要向上平移，直线向上平移的话是增加截距，也就是说应该写作$\boldsymbol{w}\cdot\boldsymbol{x}+b+一个正数=0$，那k在等号的右边，怎么可能是一个大于0的数呢？同理，向下平移的话应该是截距减小，所以k'也不可能是一个小于0的数。所以p和r的符号，不完全是平移的结果。

有人说，“直线以上的点带入直线为正，直线以下的点带入直线为负”是直线的性质，这又是另一种误解。假设我们有穿过圆点的直线$y=x$，我们取点(x,y) = (0,1)这个在直线上的点为例，如果直线的表达式写作$y-x=0$，则点(0,1)带入后为正1，如果我们将直线的表达式写作$x-y=0$，则带入(0,1)后结果为-1。所以，一个点在直线的上方，究竟会返回什么样的符号，是跟直线的表达式的写法有关的，不是直线上的点都为正，直线下的点都为负。

可能细心的小伙伴会发现，我们规定了k和k'的符号与标签的符号一致，所以有人会说，k和k'的符号，由所代表的点的标签的符号决定。这不是完全错误的，但这种说法无法解释，为什么我们就可以这样规定。并且，标签可以不是{-1,1}，可以是{0, 1}，可以是{1,2}，两个标签之间并不需要是彼此的负数，标签的取值其实也是我们规定的。

那k和k'的符号，到底是依据什么来定的呢？数学中很多过程，都是可以取巧的，来看以下过程。

记得我们的决策边界如果写成矩阵，可以表示为：

$$[a,-1]*\begin{bmatrix}x_2\\x_1\end{bmatrix}+b=0$$
$$\boldsymbol{w}\cdot\boldsymbol{x}+b=0$$

紫色点x_p毫无疑问是在决策边界的上方的，此时我将决策边界向上移动，形成一条过x_p的直线。根据我们平移的规则，直线向上平移，是在截距后加一个正数，则等号的右边是一个负数，假设这个数等于-3，则有：

$$[a,-1]*\begin{bmatrix}x_2\\x_1\end{bmatrix}+b=-3$$

另等式两边同时乘以 -1:

$$[-a,1]*\begin{bmatrix}x_2\\x_1\end{bmatrix}+(-b)=3$$
$$\boldsymbol{w}\cdot\boldsymbol{x}+b=3$$

可以注意到，我们的参数向量由[a,-1]变成了[-a,1]，b变成了$-b$，但参数向量依旧可以被表示成$\boldsymbol{w}$，只是它是原来的负数了，截距依旧可以被表示成b，只是如果它原来是正，它现在就是负数了，如果它原本就是负数，那它现在就是正数了。在这个调整中，我们通过将向上平移时产生的负号放入了参数向量和截距当中，这不影响我们求解，只不过我们求解出的参数向量和截距的符号变化了，但决策边界本身没有变化。所以我们依然可以使用原来的字母来表示这些更新后的参数向量和截距。通过这种方法，我们让$\boldsymbol{w}\cdot\boldsymbol{x}+b=k$中的k大于0。我们让k大于0的目的，是为了它的符号能够与我们的标签的符号一致，都是为了后续计算和推导的简便。

为了推导和计算的简便，我们规定：

标签是{-1,1}
决策边界以上的点，标签都为正，并且通过调整w和b的符号，让这个点在$w \cdot x + b$上得出的结果为正。
决策边界以下的点，标签都为负，并且通过调整w和b的符号，让这个点在$w \cdot x + b$上得出的结果为负。
结论：决策边界以上的点都为正，以下的点都为负，是我们为了计算简便，而人为规定的。这种规定，不会影响对参数向量w和截距b的求解。

有了这个理解，剩下的推导就简单多了。我们之前说过，决策边界的两边要有两个超平面，这两各超平面在二维空间中就是两条平行线，而他们之间的距离就是我们的边际d。这而决策边界位于这两条线的中间，所以这两条平行线必然是对称的。我们另这两条平行线被表示为：

$$w \cdot x + b = k$$
$$w \cdot x + b = -k$$

两个表达式同时除以k，则可以得到：

$$w \cdot x + b = 1$$
$$w \cdot x + b = -1$$

这就是我们平行于决策边界的两条线的表达式。此时，我们可以让这两条线分别过**两类数据中距离我们的决策边界最近的点，这些点就被称为"支持向量"**，而决策边界永远在这两条线的中间，所以可以被调整。我们另紫色类的点为x_p，红色类的点为x_r，则我们可以得到：

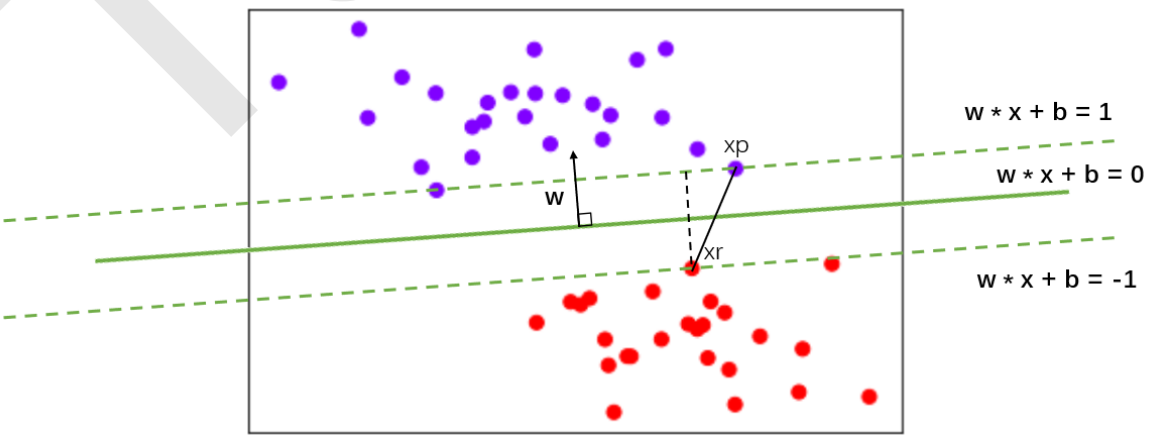
$$w \cdot x_p + b = 1$$
$$w \cdot x_r + b = -1$$

两个式子相减，则有：

$$w \cdot (x_p - x_r) = 2$$

如下图所示，$(x_p - x_r)$可表示为两点之间的连线，而我们的边际d是平行于w的，所以我们现在，相当于是得到了三角型中的斜边，并且知道一条直角边的方向。在线性代数中，向量有这样的性质：向量a除以向量b的模长$||b||$，可以得到向量a在向量b的方向上的投影的长度。所以，我们另上述式子两边同时除以$||w||$，则可以得到：

$$\frac{w \cdot (x_p - x_r)}{||w||} = \frac{2}{||w||}$$
$$\therefore d = \frac{2}{||w||}$$



还记得我们想求什么吗？最大边界所对应的决策边界，那问题就简单了，**要最大化d，就求解w的最小值。**极值问题可以相互转化，我们可以把求解w的最小值转化为，求解以下函数的最小值：

$$f(w) = \frac{||w||^2}{2}$$

只所以要在模长上加上平方，是因为模长的本质是一个距离，所以它是一个带根号的存在，我们对它取平方，是为了消除根号（其实模长的本质是向量w的l2范式，还记得l2范式公式如何写的小伙伴必定豁然开朗）。

我们的两条虚线表示的超平面，是数据边缘所在的点。所以对于任意样本i，我们可以把决策函数写作：

$$\boldsymbol{w} \cdot \boldsymbol{x_i} + b \geq 1 \quad if \ \ y_i = 1$$
$$\boldsymbol{w} \cdot \boldsymbol{x_i} + b \leq -1 \quad if \ \ y_i = -1$$

整理一下，我们可以把两个式子整合成：

$$y_i(\boldsymbol{w} \cdot \boldsymbol{x_i} + b) \geq 1, i = 1, 2....N$$

于是，我们就得到了我们SVM的损失函数：

$$\min_{w,b} \frac{||\boldsymbol{w}||^2}{2}$$
$$subject\ to\ y_i(\boldsymbol{w} \cdot \boldsymbol{x_i} + b \geq 1),$$
$$i = 1, 2, \ldots N.$$

到这里，我们就完成了，对SVM第一层理解的第一部分：线性SVM做二分类的损失函数。我们最小化这个损失函数，来求解w的值。

2.1.2 【完整版】用拉格朗日对偶函数求解线性SVM

2.1.3 线性SVM决策过程的可视化

我们可以使用sklearn中的式子来为可视化我们的决策边界，和决策边界平行的两个超平面。

1. **导入需要的模块**

```
from sklearn.datasets import make_blobs
from sklearn.svm import SVC
import matplotlib.pyplot as plt
import numpy as np
```

2. **实例化数据集，可视化数据集**

```
X,y = make_blobs(n_samples=50, centers=2, random_state=0,cluster_std=0.6)
plt.scatter(X[:,0],X[:,1],c=y,s=50,cmap="rainbow")
plt.xticks([])
plt.yticks([])
plt.show()
```

3. **定义画决策边界的函数**

```
#首先要有散点图
plt.scatter(X[:,0],X[:,1],c=y,s=50,cmap="rainbow")

ax = plt.gca() #获取当前的子图，如果不存在，则创建新的子图
xlim = ax.get_xlim()
ylim = ax.get_ylim() #默认创建(0.0, 1.0)范围内的横纵坐标

#要画决策边界，必须要有网格
axisx = np.linspace(xlim[0],xlim[1],30)
axisy = np.linspace(ylim[0],ylim[1],30)
axisy,axisx = np.meshgrid(axisy,axisx)
#将特征向量转换为特征矩阵的函数
#核心是将两个特征向量广播，以便获取y.shape * x.shape这么多个坐标点的横坐标和纵坐标
xy = np.vstack([axisx.ravel(), axisy.ravel()]).T
#获取y.shape * x.shape这么多个坐标点
#其中ravel()是降维函数，vstack能够将多个结构一致的一维数组按行堆叠起来
#xy就是已经形成的网络，它是遍布在整个画布上的密集的点

a = np.array([1,2,3])
b = np.array([7,8])
#两两组合，会得到多少个坐标?
#答案是6个，分别是 (1,7),(2,7),(3,7),(1,8),(2,8),(3,8)

v1,v2 = np.meshgrid(a,b)

v1

v2

v = np.vstack([v1.ravel(), v2.ravel()]).T

#建模，通过fit计算出对应的决策边界
clf = SVC(kernel = "linear").fit(X,y)
P = clf.decision_function(xy).reshape(axisx.shape)
#重要接口decision_function，返回每个输入的样本所对应的到决策边界的距离
#然后再将这个距离转换为axisx的结构

#画决策边界和平行于决策边界的超平面
ax.contour(axisx,axisy,P
           ,colors="k"
           ,levels=[-1,0,1]
           ,alpha=0.5
           ,linestyles=["--","-","--"])

ax.set_xlim(xlim)
ax.set_ylim(ylim)

#将上述过程包装成函数:
def plot_svc_decision_function(model,ax=None):
    if ax is None:
        ax = plt.gca()
    xlim = ax.get_xlim()
    ylim = ax.get_ylim()
```

```
    x = np.linspace(xlim[0],xlim[1],30)
    y = np.linspace(ylim[0],ylim[1],30)
    Y,X = np.meshgrid(y,x)
    xy = np.vstack([X.ravel(), Y.ravel()]).T
    P = model.decision_function(xy).reshape(X.shape)

    ax.contour(X, Y, P,colors="k",levels=[-1,0,1],alpha=0.5,linestyles=["--","-","--"])
    ax.set_xlim(xlim)
    ax.set_ylim(ylim)

#则可以写作:
clf = SVC(kernel = "linear").fit(X,y)
plt.scatter(X[:,0],X[:,1],c=y,s=50,cmap="rainbow")
plot_svc_decision_function(clf)
```

我们目前所有的例子，都是基于数据是线性可分的状况来说明的。如果数据是线性不可分呢？比如说：

```
from sklearn.datasets import make_circles
X,y = make_circles(100, factor=0.1, noise=.1)

X.shape

y.shape

plt.scatter(X[:,0],X[:,1],c=y,s=50,cmap="rainbow")
plt.show()
```

试试看用我们已经定义的函数来划分这个数据的决策边界：

```
clf = SVC(kernel = "linear").fit(X,y)
plt.scatter(X[:,0],X[:,1],c=y,s=50,cmap="rainbow")
plot_svc_decision_function(clf)
```

明显，现在线性SVM已经不适合于我们的状况了，我们无法找出一条直线来划分我们的数据集，让直线的两边分别是两种类别。这个时候，如果我们能够在原本的X和y的基础上，添加一个维度r，变成三维，我们可视化这个数据，来看看添加维度让我们的数据如何变化。

```
#定义一个由x计算出来的新维度r
r = np.exp(-(X**2).sum(1))

rlim = np.linspace(min(r),max(r),0.2)

from mpl_toolkits import mplot3d

#定义一个绘制三维图像的函数
#elev表示上下旋转的角度
#azim表示平行旋转的角度
def plot_3D(elev=30,azim=30,X=X,y=y):
    ax = plt.subplot(projection="3d")
    ax.scatter3D(X[:,0],X[:,1],r,c=y,s=50,cmap='rainbow')
```

```
    ax.view_init(elev=elev,azim=azim)
    ax.set_xlabel("x")
    ax.set_ylabel("y")
    ax.set_zlabel("r")
    plt.show()

plot_3D()

#如果放到jupyter notebook中运行
from sklearn.svm import SVC
import matplotlib.pyplot as plt
import numpy as np

from sklearn.datasets import make_circles
X,y = make_circles(100, factor=0.1, noise=.1)
plt.scatter(X[:,0],X[:,1],c=y,s=50,cmap="rainbow")

def plot_svc_decision_function(model,ax=None):
    if ax is None:
        ax = plt.gca()
    xlim = ax.get_xlim()
    ylim = ax.get_ylim()

    x = np.linspace(xlim[0],xlim[1],30)
    y = np.linspace(ylim[0],ylim[1],30)
    Y,X = np.meshgrid(y,x)
    xy = np.vstack([X.ravel(), Y.ravel()]).T
    P = model.decision_function(xy).reshape(X.shape)

    ax.contour(X, Y, P,colors="k",levels=[-1,0,1],alpha=0.5,linestyles=["--","-","--"])
    ax.set_xlim(xlim)
    ax.set_ylim(ylim)

clf = SVC(kernel = "linear").fit(X,y)
plt.scatter(X[:,0],X[:,1],c=y,s=50,cmap="rainbow")
plot_svc_decision_function(clf)

r = np.exp(-(X**2).sum(1))

rlim = np.linspace(min(r),max(r),0.2)

from mpl_toolkits import mplot3d

def plot_3D(elev=30,azim=30,X=X,y=y):
    ax = plt.subplot(projection="3d")
    ax.scatter3D(X[:,0],X[:,1],r,c=y,s=50,cmap='rainbow')
    ax.view_init(elev=elev,azim=azim)
    ax.set_xlabel("x")
    ax.set_ylabel("y")
    ax.set_zlabel("r")
    plt.show()

from ipywidgets import interact,fixed
```

```
interact(plot_3D,elev=[0,30],azip=(-180,180),X=fixed(X),y=fixed(y))
plt.show()
```

此时我们的数据在三维空间中，我们的超平面就是一个二维平面。明显我们可以用一个平面将两类数据隔开，这个平面就是我们的超平面。我们刚才做的，计算r，并将r作为数据的第三维度来讲数据升维的过程，被称为“核变换”，即是将数据投影到高维空间中，以寻找能够将数据完美分割的超平面，而在高维空间中计算来找出超平面的函数就叫做核函数。在SVM中，这个功能由参数“kernel”控制。之前我们一直使用这个参数，但是没有给大家解释，我们使用的是“linear"，线性核函数，只能用于线性的情况。刚才我们使用的计算r的方法，其实是高斯径向基核函数，在参数”kernel“中输入”rbf“就可以使用。我们来看看模型找出的决策边界时什么样：

```
clf = SVC(kernel = "rbf").fit(X,y)
plt.scatter(X[:,0],X[:,1],c=y,s=50,cmap="rainbow")
plot_svc_decision_function(clf)
```

核函数是我们处理非线性问题的关键。

2.3【完整版】非线性SVM与核函数

2.3.1【完整版】重要参数kernel & degree & gamma

线性核：

$$k(x,y) = x^T y = \langle x,y \rangle$$

多项式核：

$$k(x,y) = (\gamma\langle x,y \rangle + r)^d$$

Sigmoid（Hyperbolic Tangent）核函数：

$$k(x,y) = tanh(\gamma\langle x,y \rangle + r)$$

高斯径向基（rbf）函数：

$$K(x,y) = e^{-\gamma||x-y||^2}, \gamma > 0$$

$$在高斯核函数中\ \gamma = \frac{1}{(2\sigma)^2}$$

2.3.2【完整版】拉格朗日对偶函数在非线性SVM上的推广

2.3【完整版】硬间隔与软间隔：重要参数C

2.3.1【完整版】SVM在软间隔数据上的推广

2.3.2【完整版】重要参数C

2.4【完整版】SVC处理多分类问题：重要参数decision_function_shape

下周内容

sklearn中的支持向量机SVM（二）

3 sklearn.svm.SVR

3.1 重要参数

3.2 重要属性和接口

4 SVR案例：线性与非线性核下的支持向量机回归

5 附录

5.1 SVR参数列表

5.2 SVR属性列表

5.3 SVR接口列表