菜菜的scikit-learn课堂08

sklearn中的支持向量机SVM - 案例直播课

小伙伴们晚上好~o(￣▽￣)ブ

我是菜菜，这里是我的sklearn课堂第八期，今晚的直播内容是支持向量机 - 案例直播课！

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

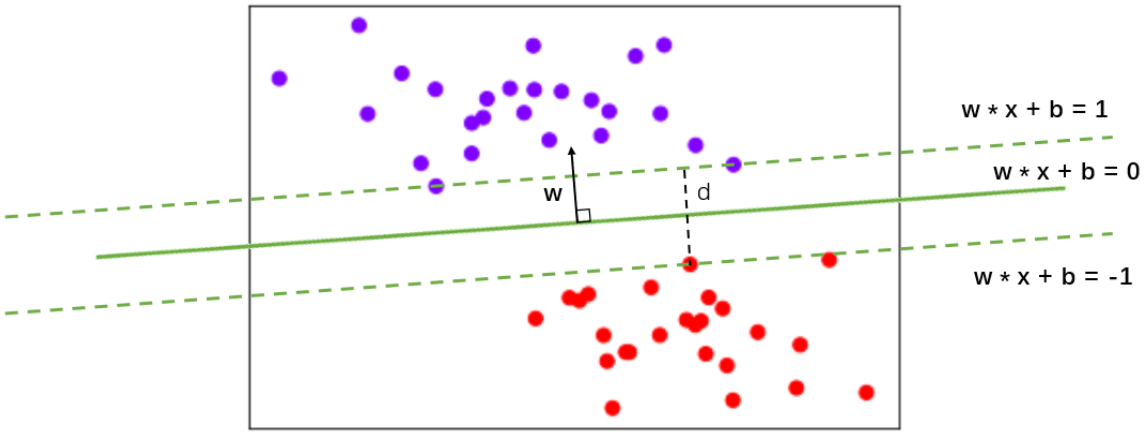
Numpy 1.15.4, **Pandas** 0.23.4, **Matplotlib** 3.0.2, **SciPy** 1.1.0

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~**

1 原理概述

在上周的SVM课程中，我们为大家介绍了支持向量机的原理，为大家推导了支持向量机的损失函数，拉格朗日函数，拉格朗日对偶函数，预测函数以及这些函数在非线性，软间隔这些情况上的推广，并且引出了核函数这个关键概念。

今天，我们将基于我们已经学过的理论，开始对核函数性质的探索。考虑到我们之前用了大概25页PDF来描述支持向量机的原理，开始今天的案例之前，我们先来简单回忆一下支持向量机是如何工作的。



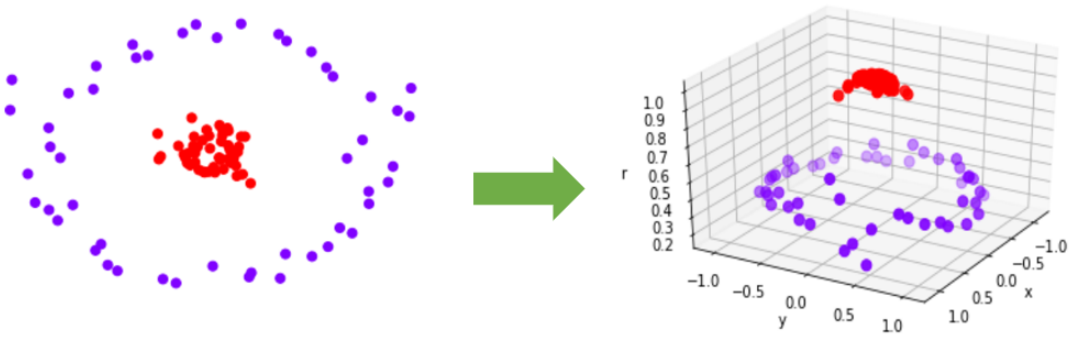
支持向量机分类器，是在数据空间中找出一个超平面作为决策边界，利用这个决策边界来对数据进行分类，并使分类误差尽量小的模型。决策边界是比所在数据空间小一维的空间，在三维数据空间中就是一个平面，在二维数据空间中就是一条直线。以二维数据为例，图中的数据集有两个特征，标签有两类，一类为紫色，一类为红色。对于这组数据，我们找出的决策边界被表达为$\boldsymbol{w} \cdot \boldsymbol{x} + b = 0$，决策边界把平面分成了上下两部分，决策边界以上的样本都分为一类，决策边界以下的样本被分为另一类。以我们的图像为例，绿色实现上部分为一类（全部都是紫色点），下部分为另一类（全都是红色点）。

平行于决策边界的两条虚线是距离决策边界相对距离为1的超平面，他们分别压过两类样本中距离决策边界最近的样本点，这些样本点就被成为支持向量。两条虚线超平面之间的距离叫做边际，简写为d。支持向量机分类器，就是以找出**最大化的边际d**为目标来求解损失函数，以求解出参数$\boldsymbol{w}$和b，以构建决策边界，然后用决策边界来分类的分类器。在这种最简单的情况下，我们的决策函数为：

$$f(x_{test}) = sign(\boldsymbol{w} \cdot \boldsymbol{x_{test}} + b) = sign(\sum_{i=1}^{N} \alpha_i y_i \boldsymbol{x_i} \cdot \boldsymbol{x_{test}} + b)$$

其中$sign(h)$是$h > 0$时返回正1，$h < 0$时返回-1的符号函数，α_i是拉格朗日乘数，y_i是样本i的真实标签，x_i是测试训练，x_{test}是测试样本，$\boldsymbol{x_i} \cdot \boldsymbol{x_{test}}$是原特征向量与测试特征向量的点积。当函数返回1，则行本i被分为正类。当函数返回-1，则样本被分为负类。

当然，不是所有数据都是线性可分的，不是所有数据我们都能够一眼看出，有一条直线，或一个平面，甚至一个超平面可以将数据完全分开。比如下面的环形数据。对于这样的数据，我们需要对它进行一个升维变化，来数据从原始的空间$\boldsymbol{x}$投射到新空间$\boldsymbol{\Phi(x)}$中。升维之后，我们明显可以找出一个平面，能够将数据切分开来。$\boldsymbol{\Phi}$是一个映射函数，它代表了某种能够将数据升维的非线性的变换，我们对数据进行这样的变换，确保数据在自己的空间中一定能够线性可分。



能够处理非线性问题的SVM的决策函数如下：

$$f(x_{test}) = sign(\boldsymbol{w} \cdot \boldsymbol{\Phi}(\boldsymbol{x_{test}}) + b) = sign(\sum_{i=1}^{N} \alpha_i y_i \boldsymbol{\Phi}(\boldsymbol{x_i}) \cdot \boldsymbol{\Phi}(\boldsymbol{x_{test}}) + b)$$

可见，除了我们对原始数据进行了升维处理之外，其他表达都和普通的线性SVM一模一样。

这种变换非常巧妙，但也带有一些实现问题。 首先，我们可能不清楚应该什么样的数据应该使用什么类型的映射函数来确保可以在变换空间中找出线性决策边界。极端情况下，数据可能会被映射到无限维度的空间中，这种高维空间可能不是那么友好，维度越多，推导和计算的难度都会随之暴增。其次，即使已知适当的映射函数，我们想要计算类似于$\boldsymbol{\Phi}(\boldsymbol{x_i}) \cdot \boldsymbol{\Phi}(\boldsymbol{x_{test}})$这样的点积，计算量可能会无比巨大，要找出超平面所付出的代价是非常昂贵的。

关键概念：核函数

而解决这些问题的数学方式，叫做“核技巧”(Kernel Trick)，是一种能够使用数据原始空间中的向量计算来表示升维后的空间中的点积结果的数学方式。具体表现为，$K(\boldsymbol{u}, \boldsymbol{v}) = \boldsymbol{\Phi}(\boldsymbol{u}) \cdot \boldsymbol{\Phi}(\boldsymbol{v})$。而这个原始空间中的点积函数$K(\boldsymbol{u}, \boldsymbol{v})$，就被叫做“核函数”(Kernel Function)。

有了核函数，我们可以将决策函数表示为：

$$f(x_{test}) = sign(\boldsymbol{w} \cdot \boldsymbol{\Phi}(\boldsymbol{x_{test}}) + b) = sign(\sum_{i=1}^{N} \alpha_i y_i K(\boldsymbol{x}, \boldsymbol{x_{test}}) + b)$$

核函数能够帮助我们解决三个问题：

第一，有了核函数之后，我们无需去担心$\boldsymbol{\Phi}$究竟应该是什么样，因为非线性SVM中的核函数都是正定核函数(positive definite kernel functions)，他们都满足美世定律(Mercer's theorem)，确保了高维空间中任意两个向量的点积一定可以被低维空间中的这两个向量的某种计算来表示（多数时候是点积的某种变换）。

第二，使用核函数计算低维度中的向量关系比计算原本的$\boldsymbol{\Phi}(\boldsymbol{x_i}) \cdot \boldsymbol{\Phi}(\boldsymbol{x_{test}})$要简单太多了。

第三，因为计算是在原始空间中进行，所以避免了维度诅咒的问题。

选用不同的核函数，就可以解决不同数据分布下的寻找超平面问题。在sklearn的SVC中，这个功能由参数“kernel”(ˈkərnl)和一系列与核函数相关的参数来进行控制。今天，我们就在乳腺癌数据集上，来探索一下各种核函数的功能和选择。

2 SVC的重要参数kernel

先来复习以下我们的SVC类：

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)*

在sklearn中实现SVC的基本流程：

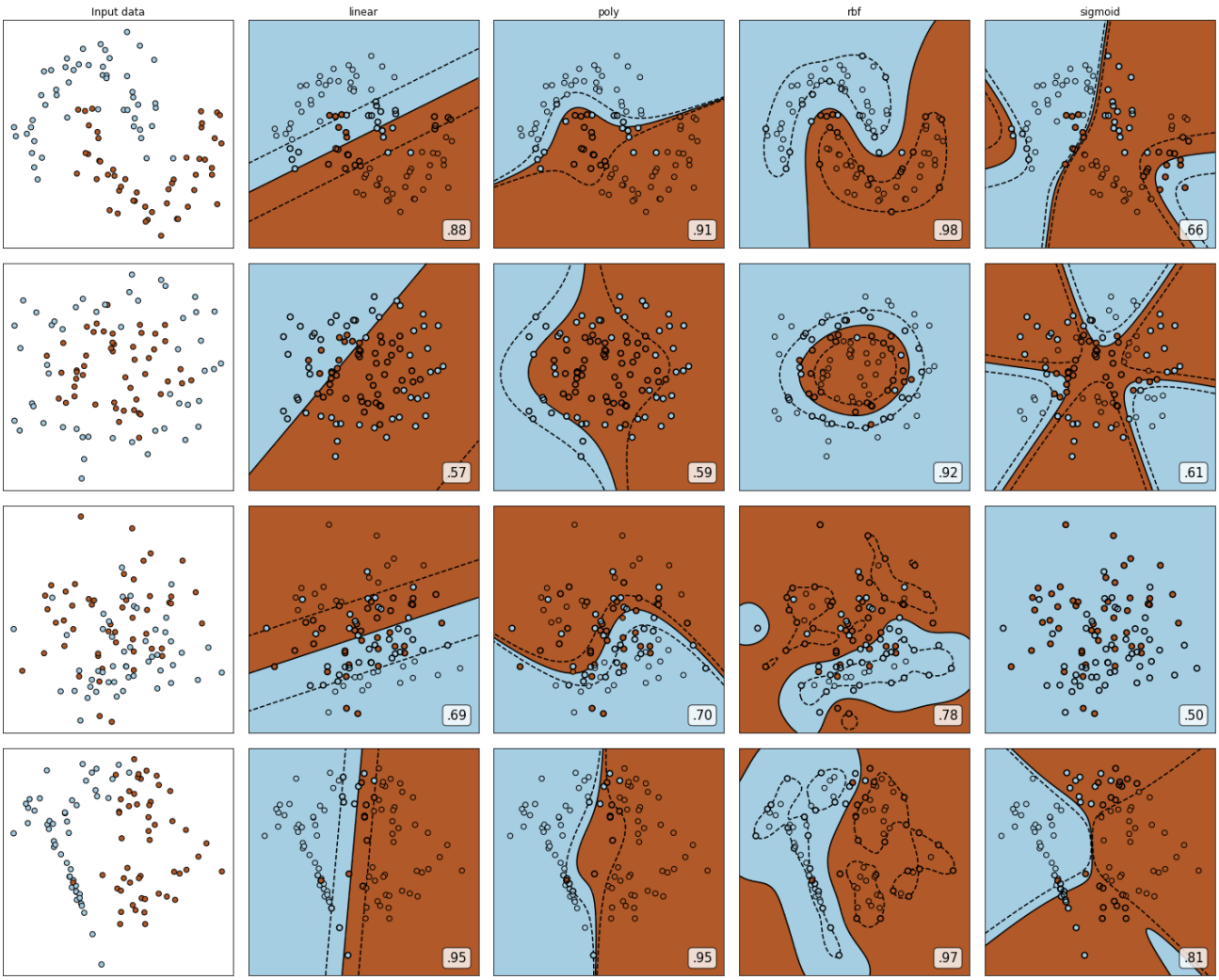
```
from sklearn.svm import SVC                          #导入需要的模块

clf = SVC()                                          #实例化
clf = clf.fit(X_train,y_train)                       #用训练集数据训练模型
result = clf.score(X_test,y_test)                    #导入测试集，从接口中调用需要的信息
```

作为SVC类最重要的参数之一，“kernel"在sklearn中可选以下几种选项：

输入	含义	解决问题	核函数的表达式	参数gamma	参数degree	参数coef0
“linear”	线性核	线性	$K(x,y)=x^Ty=x\cdot y$	No	No	No
“poly"	多项式核	偏线性	$K(x,y)=(\gamma(x\cdot y)+r)^d$	Yes	Yes	Yes
"sigmoid"	双曲正切核	非线性	$K(x,y)=tanh(\gamma(x\cdot y)+r)$	Yes	No	Yes
"rbf"	高斯径向基	偏非线性	$K(x,y)=e^{-\gamma\|x-y\|^2},\gamma>0$	Yes	No	No

可以看出，除了选项"linear"之外，其他核函数都可以处理非线性问题。多项式核函数有次数d，当d为1的时候它就是再处理线性问题，当d为更高次项的时候它就是在处理非线性问题。那究竟什么时候选择哪一个核函数呢？遗憾的是，关于核函数在不同数据集上的研究甚少，谷歌学术上的论文中也没有几篇是研究核函数在SVM中的运用的，更多的是关于核函数在深度学习，神经网络中如何使用。在sklearn中，也没有提供任何关于如何选取核函数的信息。

但无论如何，我们还是可以通过在不同的核函数中循环去找寻最佳的核函数来对核函数进行一个选取。我创造了一系列线性或非线性可分的数据，绘制出每个数据集上SVC在不同核函数下的决策边界，并计算SVC在不同核函数下分类准确率来观察核函数的效用。这个图是如何绘制出来的，在我们完整的付费课程中大家可以看到完整的代码。要讲解绘制这个图像的代码就需要40分钟，所以在今天一小时的直播中，我就直接拿它作为结论来用了：



可以观察到，线性核函数和多项式核函数在非线性数据上表现会浮动，如果数据相对线性可分，则表现不错，如果是像环形数据那样彻底不可分的，则表现糟糕。在线性数据集上，线性核函数和多项式核函数即便有扰动项也可以表现不错，可见多项式核函数是虽然也可以处理非线性情况，但更偏向于线性的功能。

Sigmoid核函数就比较尴尬了，它在非线性数据上强于两个线性核函数，但效果明显不如rbf，它在线性数据上完全比不上线性的核函数们，对扰动项的抵抗也比较弱，所以它功能比较弱小，很少被用到。

rbf，高斯径向基核函数基本在任何数据集上都表现不错，属于比较万能的核函数。我个人的经验是，无论如何先试试看高斯径向基核函数，它适用于核转换到很高的空间的情况，在各种情况下往往效果都很不错，如果rbf效果不好，那我们再试试看其他的核函数。另外，多项式核函数多被用于图像处理之中。

3 乳腺癌数据集下探索核函数的性质

3.1 探索kernel该如何选取

看起来，除了Sigmoid核函数，其他核函数效果都还不错。但其实各个核函数都有自己的问题。接下来，我们就使用乳腺癌数据集作为例子来展示一下：

```
from sklearn.datasets import load_breast_cancer
from sklearn.svm import SVC
```

```
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import numpy as np
from time import time
import datetime

data = load_breast_cancer()
X = data.data
y = data.target

X.shape
np.unique(y)

plt.scatter(X[:,0],X[:,1],c=y)
plt.show()

Xtrain, Xtest, Ytrain, Ytest = train_test_split(X,y,test_size=0.3,random_state=420)

Kernel = ["linear","poly","rbf","sigmoid"]

for kernel in Kernel:
    time0 = time()
    clf= SVC(kernel = kernel
             , gamma="auto"
            # , degree = 1
             , cache_size=5000
            ).fit(Xtrain,Ytrain)
    print("The accuracy under kernel %s is %f" % (kernel,clf.score(Xtest,Ytest)))
    print(datetime.datetime.fromtimestamp(time()-time0).strftime("%M:%S:%f"))
```

然后我们发现，怎么跑都跑不出来。模型一直停留在线性核函数之后，就没有再打印结果了。这证明，多项式核函数此时此刻要消耗大量的时间，运算非常的缓慢。让我们在循环中去掉多项式核函数，再试试看能否跑出结果：

```
Kernel = ["linear","rbf","sigmoid"]

for kernel in Kernel:
    time0 = time()
    clf= SVC(kernel = kernel
             , gamma="auto"
            # , degree = 1
             , cache_size=5000
            ).fit(Xtrain,Ytrain)
    print("The accuracy under kernel %s is %f" % (kernel,clf.score(Xtest,Ytest)))
    print(datetime.datetime.fromtimestamp(time()-time0).strftime("%M:%S:%f"))
```

我们可以有两个发现。首先，乳腺癌数据集是一个线性数据集，线性核函数跑出来的效果很好。rbf和sigmoid两个擅长非线性的数据从效果上来看完全不可用。其次，线性核函数的运行速度远远不如非线性的两个核函数。

如果数据是线性的，那如果我们把degree参数调整为1，多项式核函数应该也可以得到不错的结果：

```
Kernel = ["linear","poly","rbf","sigmoid"]

for kernel in Kernel:
    time0 = time()
    clf= SVC(kernel = kernel
             , gamma="auto"
             , degree = 1
             , cache_size=5000
            ).fit(Xtrain,Ytrain)
    print("The accuracy under kernel %s is %f" % (kernel,clf.score(Xtest,Ytest)))
    print(datetime.datetime.fromtimestamp(time()-time0).strftime("%M:%S:%f"))
```

多项式核函数的运行速度立刻加快了，并且精度也提升到了接近线性核函数的水平，可喜可贺。但是，我们之前的实验中，我们了解说，rbf在线性数据上也可以表现得非常好，那在这里，为什么跑出来的结果如此糟糕呢？

其实，这里真正的问题是数据的量纲问题。回忆一下我们如何求解决策边界，如何判断点是否在决策边界的一边？是靠计算"距离"，虽然我们不能说SVM是完全的距离类模型，但是它严重受到数据量纲的影响。让我们来探索一下乳腺癌数据集的量纲：

```
import pandas as pd
data = pd.DataFrame(X)
data.describe([0.01,0.05,0.1,0.25,0.5,0.75,0.9,0.99]).T
```

一眼望去，果然数据存在严重的量纲不一的问题。我们来使用数据预处理中的标准化的类，对数据进行标准化：

```
from sklearn.preprocessing import StandardScaler
X = StandardScaler().fit_transform(X)
data = pd.DataFrame(X)
data.describe([0.01,0.05,0.1,0.25,0.5,0.75,0.9,0.99]).T
```

标准化完毕后，再次让SVC在核函数中遍历，此时我们把degree的数值设定为1，观察各个核函数在去量纲后的数据上的表现：

```
Xtrain, Xtest, Ytrain, Ytest = train_test_split(X,y,test_size=0.3,random_state=420)

Kernel = ["linear","poly","rbf","sigmoid"]

for kernel in Kernel:
    time0 = time()
    clf= SVC(kernel = kernel
             , gamma="auto"
             , degree = 1
             , cache_size=5000
            ).fit(Xtrain,Ytrain)
    print("The accuracy under kernel %s is %f" % (kernel,clf.score(Xtest,Ytest)))
    print(datetime.datetime.fromtimestamp(time()-time0).strftime("%M:%S:%f"))
```

量纲统一之后，可以观察到，所有核函数的运算时间都大大地减少了，尤其是对于线性核来说，而多项式核函数居然变成了计算最快的。其次，rbf表现出了非常优秀的结果。经过我们的探索，我们可以得到的结论是：

1. **线性核，尤其是多项式核函数在高次项时计算非常缓慢**

2. **rbf和多项式核函数都不擅长处理量纲不统一的数据集**

幸运的是，这两个缺点都可以由数据无量纲化来解决。因此，**SVM执行之前，非常推荐先进行数据的无量纲化**！到了这一步，我们是否已经完成建模了呢？虽然线性核函数的效果是最好的，但它是没有核函数相关参数可以调整的，rbf和多项式却还有着可以调整的相关参数，接下来我们就来看看这些参数。

3.2 选取与核函数相关的参数：degree & gamma & coef0

输入	含义	解决问题	核函数的表达式	参数gamma	参数degree	参数coef0
“linear”	线性核	线性	$K(x,y)=x^Ty=x\cdot y$	No	No	No
“poly"	多项式核	偏线性	$K(x,y)=(\gamma(x\cdot y)+r)^d$	Yes	Yes	Yes
"sigmoid"	双曲正切核	非线性	$K(x,y)=tanh(\gamma(x\cdot y)+r)$	Yes	No	Yes
"rbf"	高斯径向基	偏非线性	$K(x,y)=e^{-\gamma\|x-y\|^2},\gamma>0$	Yes	No	No

在知道如何选取核函数后，我们还要观察一下除了kernel之外的核函数相关的参数。对于线性核函数，"kernel"是唯一能够影响它的参数，但是对于其他三种非线性核函数，他们还受到参数gamma，degreel以及coef0的影响。参数gamma就是表达式中的γ，degree就是多项式核函数的次数d，参数coef0就是常数项r。其中，高斯径向基核函数受到gamma的影响，而多项式核函数受到全部三个参数的影响。

参数	含义
degree	整数，可不填，默认3 多项式核函数的次数（'poly'），如果核函数没有选择"poly"，这个参数会被忽略
gamma	浮点数，可不填，默认“auto" 核函数的系数，仅在参数Kernel的选项为”rbf","poly"和"sigmoid”的时候有效 输入“auto"，自动使用1/(n_features)作为gamma的取值 输入"scale"，则使用1/(n_features * X.std())作为gamma的取值 输入"auto_deprecated"，则表示没有传递明确的gamma值（不推荐使用）
coef0	浮点数，可不填，默认=0.0 核函数中的常数项，它只在参数kernel为'poly'和'sigmoid'的时候有效。

但从核函数的公式来看，我们其实很难去界定具体每个参数如何影响了SVM的表现。当gamma的符号变化，或者degree的大小变化时，核函数本身甚至都不是永远单调的。所以如果我们想要彻底地理解这三个参数，我们要先推导出它们如何影响核函数地变化，再找出核函数的变化如何影响了我们的预测函数（可能改变我们的核变化所在的维度），再判断出决策边界随着预测函数的改变发生了怎样的变化。无论是从数学的角度来说还是从实践的角度来说，这个过程太复杂也太低效。所以，我们往往避免去真正探究这些参数如何影响了我们的核函数，而直接使用学习曲线或者网格搜索来帮助我们查找最佳的参数组合。

对于高斯径向基核函数，调整gamma的方式其实比较容易，那就是画学习曲线。我们来试试看高斯径向基核函数rbf的参数gamma在乳腺癌数据集上的表现：

```
score = []
gamma_range = np.logspace(-10, 1, 50) #返回在对数刻度上均匀间隔的数字
for i in gamma_range:
    clf = SVC(kernel="rbf",gamma = i,cache_size=5000).fit(Xtrain,Ytrain)
    score.append(clf.score(Xtest,Ytest))

print(max(score), gamma_range[score.index(max(score))])
plt.plot(gamma_range,score)
plt.show()
```

通过学习曲线，很容就找出了rbf的最佳gamma值。但我们观察到，这其实与线性核函数的准确率一模一样之前的准确率。我们可以多次调整gamma_range来观察结果，可以发现97.6608应该是rbf核函数的极限了。

但对于多项式核函数来说，一切就没有那么容易了，因为三个参数共同作用在一个数学公式上影响它的效果，因此我们往往使用网格搜索来共同调整三个对多项式核函数有影响的参数。依然使用乳腺癌数据集。

```
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.model_selection import GridSearchCV

time0 = time()

gamma_range = np.logspace(-10,1,20)
coef0_range = np.linspace(0,5,10)

param_grid = dict(gamma = gamma_range
                  ,coef0 = coef0_range)

cv = StratifiedShuffleSplit(n_splits=5, test_size=0.3, random_state=420)
grid = GridSearchCV(SVC(kernel = "poly",degree=1,cache_size=5000),
param_grid=param_grid, cv=cv)
grid.fit(X, y)

print("The best parameters are %s with a score of %0.5f" % (grid.best_params_,
grid.best_score_))
print(datetime.datetime.fromtimestamp(time()-time0).strftime("%M:%S:%f"))
```

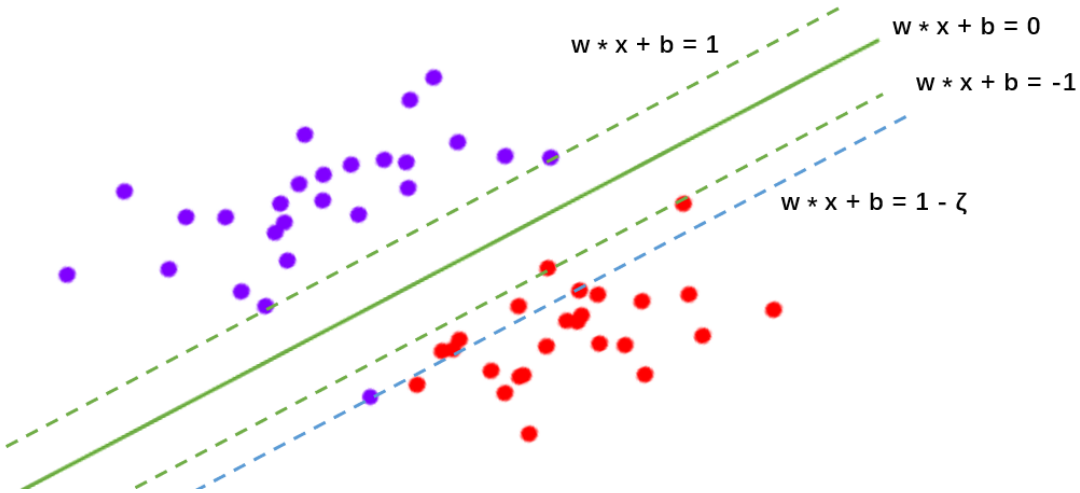
可以发现，网格搜索为我们返回了参数coef0=0，gamma=0.18329807108324375，但整体的分数是0.96959，虽然比调参前略有提高，但依然没有超过线性核函数核rbf的结果。可见，如果最初选择核函数的时候，你就发现多项式的结果不如rbf和线性核函数，那就不要挣扎了，试试看调整rbf或者直接使用线性。

4 软间隔与重要参数C

当然，不是所有数据都是完全线性可分的。可能存在着一条直线能够将大部分数据点的类别划分正确，但无论如何也无法将全部的点分对，如同下图所展示的图，存在着混杂在红色类中的紫色点，这种情况下没有一条直线能够将两类数据完全分类正确。

关键概念：硬间隔与软间隔

当两组数据是完全线性可分，我们可以找出一个决策边界使得训练集上的分类误差为0，这两种数据就被称为是存在"硬间隔"的。当两组数据几乎是完全线性可分的，但决策边界在训练集上存在较小的训练误差，这两种数据就被称为是存在"软间隔"。



这个时候，我们的决策边界就不是单纯地寻求最大边际了，因为对于软间隔地数据来说，边际越大被分错的样本也就会越多，因此我们需要找出一个"最大边际"与"被分错的样本数量"之间的平衡。参数C用于权衡"训练样本的正确分类"与"决策函数的边际最大化"两个不可同时完成的目标，希望找出一个平衡点来让模型的效果最佳。

参数	含义
C	浮点数，默认1，必须大于等于0，可不填 松弛系数的惩罚项系数。如果C值设定比较大，那SVC可能会选择边际较小的，能够更好地分类所有训练点的决策边界，不过模型的训练时间也会更长。如果C的设定值较高，那SVC会尽量最大化边界，决策功能会更简单，但代价是训练的准确度。换句话说，C在SVM中的影响就像正则化参数对逻辑回归的影响。

在实际使用中，C和核函数的相关参数（gamma，degree等等）们搭配，往往是SVM调参的重点。与gamma不同，C没有在对偶函数中出现，并且是明确了调参目标的，所以我们可以明确我们究竟是否需要训练集上的高精确度来调整C的方向。默认情况下C为1，通常来说这都是一个合理的参数。 如果我们的数据很嘈杂，那我们往往减小C。当然，我们也可以使用网格搜索或者学习曲线来调整C的值。

```
#调线性核函数
score = []
C_range = np.linspace(0.01,30,50)
for i in C_range:
```

```
    clf = SVC(kernel="linear",C=i,cache_size=5000).fit(Xtrain,Ytrain)
    score.append(clf.score(Xtest,Ytest))

print(max(score), C_range[score.index(max(score))])
plt.plot(C_range,score)
plt.show()

#换rbf
score = []
C_range = np.linspace(0.01,30,50)
for i in C_range:
    clf = SVC(kernel="rbf",C=i,gamma =
0.012742749857031322,cache_size=5000).fit(Xtrain,Ytrain)
    score.append(clf.score(Xtest,Ytest))

print(max(score), C_range[score.index(max(score))])
plt.plot(C_range,score)
plt.show()

#进一步细化
score = []
C_range = np.linspace(5,7,50)
for i in C_range:
    clf = SVC(kernel="rbf",C=i,gamma =
0.012742749857031322,cache_size=5000).fit(Xtrain,Ytrain)
    score.append(clf.score(Xtest,Ytest))

print(max(score), C_range[score.index(max(score))])
plt.plot(C_range,score)
plt.show()
```

此时，我们找到了乳腺癌数据集上的最优解：rbf核函数下的98.24%的准确率。当然，我们还可以使用交叉验证来改进我们的模型，获得不同测试集和训练集上的交叉验证结果。但上述过程，为大家展现了如何选择正确的核函数，以及如何调整核函数的参数，过程虽然简单，但是希望可以对大家有所启发。

5 完整版SVM会有哪些内容?

目录

菜菜的scikit-learn课堂08

sklearn中的支持向量机SVM（下）

6 完整的12期课会有哪些内容?

菜菜的机器学习sklearn课堂

期数	直播日期	完整版上线日期	主题	案例	涉及的模块或库
01期	11月7日	11月11日	决策树	红酒数据集的分类 泰坦尼克号的幸存者预测	sklearn.tree
02期	11月14日	11月18日	随机森林	红酒数据集的分类 乳腺癌患病者的预测 & 调参	sklearn.ensemble
03期	11月21日	11月25日	数据预处理和特征工程	自制文字数据集上的数据预处理 识别手写数字数据集的特征工程	sklearn.preprocessing sklearn.feature_selection
04期	11月28日	12月2日	主成分分析PCA与奇异值分解SVD	高维数据的可视化 用人脸识别看PCA降维后的信息保存量 用降维算法做噪音过滤 PCA在识别手写数字数据集的降维	sklearn.decomposition
05期	12月5日	12月9日	逻辑回归	用逻辑回归制作评分卡（A卡）	sklearn.linear_model
06期	12月12日	12月16日	聚类算法与K-Means	自制数据集上的聚类性质探索 KMeans矢量量化颐和园图像	sklearn.cluster
07期	12月19日	12月23日	SVM（上）	乳腺癌数据集上探索核函数的效应	sklearn.svm
08期	12月26日	12月30日	SVM（下）	研发中，敬请期待	sklearn.svm
09期	1月2日	1月6日	线性回归，岭回归与Lasso	研发中，敬请期待	sklearn.linear_model
10期	1月9日	1月13日	XGBoost	研发中，敬请期待	xgboost
11期	1月16日	1月20日	朴素贝叶斯	研发中，敬请期待	sklearn.naive_bayes
12期	1月23日	1月27日	神经网络	研发中，敬请期待	sklearn.neural_network