**主成分分析（PCA）和基于核函数的主成分分析（KPCA）入门**

# 前言

主成分分析是在做特征筛选时的重要手段，这个方法在大部分的书中都只是介绍了步骤方法，并没有从头到尾把这个事情给说清楚。本文的目的是把PCA和KPCA给说清楚。主要参考了YouTube上李政轩的[Principal Component Analysis and Kernel Principal Component Analysis](https://www.youtube.com/watch?v=G2NRnh7W4NQ&t=1s)这个视频（强烈推荐看一下）。

# PCA的原理

## 什么是投影

主成分分析所做的工作就是将数据集从高维投影到低维，从而用极少的几个特征来涵盖大部分的数据集信息。   
所谓的投影，就是下图所示的这样。



图1：向量投影图

*xj*投影到*v*上的向量为

*x*′*j*=(||*xj*||*cosθ*)*v*||*v*||

其中，*θ*为*xj*与*v*的夹角。   
由于向量之间的内积为

<*xj*,*v*>=||*xj*||⋅||*v*||⋅*cosθ*

故有

*x*′*j*=<*xj*,*v*>||*v*||2*v*

如果我们把*v*设置成单位向量的话（即||*v*||=1）就有

*x*′*j*=<*xj*,*v*>*v*

也就是说我们只要求出<*xj*,*v*>就可以知道*xj*投影到*v*上的大小了。   
又由于在坐标当中，内积可以表示为

<*xj*,*v*>=*xTj*⋅*v*=*vT*⋅*xj*

故可以用*vT*⋅*xj*来表示投影后的数值大小。

## 投影后的方差

主成分分析认为，沿某特征分布的数据的方差越大，则该特征所包含的信息越多，也就是所谓的主成分。   
我们已经知道了可以用*vT*⋅*xj*

来表示投影后的数值大小，那么我们现在就可以算出投影后的方差大小了。注意我们么已经把数据标准化过了，所以*vT*⋅*x*的均值为*vT*⋅0=0

。

*σ*2=1*N*−1∑*Ni*=1(*vTxi*−0)2=1*N*−1∑*Ni*=1(*vTxi*)(*vTxi*)

注意到*vTxi*是一个数值，不是向量，故有*vTxi*=(*vTxi*)*T*于是

*σ*2=1*N*−1∑*Ni*=1*vTxixTiv*=*vT*(1*N*−1∑*Ni*=1*xixTi*)*v*=*vTCv*

其中，*C*=1*N*−1∑*Ni*=1*xixTi*是一个*m*×*m*的矩阵，*m*为特征的个数。   
好了，如果我们要找到最大的方差，也就是要找到一个向量*v*使得方差最大。

## 转化为求特征值的问题

我们可以将求最大方差的问题写成

*maxvTCvs*.*t*.||*v*||=1

又由于||*v*||=*vTv* ，故上式即

*maxvTCvs*.*t*.*vTv*=1

利用拉格朗日乘子法可以将上述问题转化为

*f*(*v*,*λ*)=*vTCv*−*λ*(*vTv*−1)

其中，*f*(*v*,*λ*)的平稳点，和我们所要求的最大方差问题是等价的，即求下述方程式的解

⎧⎩⎨⎪⎪∂*f*∂*v*=2*Cv*−2*λv*=0∂*f*∂*λ*=*vTv*−1=0

上述方程组等价于

{*Cv*=*λv*||*v*||=1

看到了没，*Cv*=*λv*不就是求特征值和特征向量的方程吗！更神奇的地方在下面，我们再回到最初求最大方差的问题

*vTCv*=*vTλv*=*λvTv*=*λ*

是不是很神奇！要求的方差就是我们这里的特征值！所以我们只需要把*Cv*=*λv*的特征值求出来，然后按大小排个序就，选出最大的几个特征值，并求出对应的特征向量，最后用这几个特征向量来完成数据集在其上的投影*vTx*，这样就完成了特征的筛选！

## 符号的表示

值得注意的是，*C*

是一个*m*×*m*

的矩阵

*C*=1*N*−1∑*Ni*=1*xixTi*=1*N*−1[*x*1,*x*2,...,*xN*]⎡⎣⎢⎢⎢⎢*xT*1*xT*2...*xTN*⎤⎦⎥⎥⎥⎥

其中，每个*xi*为一个列向量

*xi*=⎡⎣⎢⎢⎢⎢⎢*x*(1)*ix*(2)*i*...*x*(*m*)*i*⎤⎦⎥⎥⎥⎥⎥

其中，*m*为特征的个数。   
为了方便表示，我们作出如下定义

*XT*=[*x*1,*x*2,...,*xN*]

于是，*C*可以表示为

*C*=1*N*−1*XTX*

# KPCA的原理

基于核函数的主成分分析和主成分分析的步骤是一样的，只不过用核函数替代了原来的数据。这里对什么是核函数不作说明，请参考其它文章。   
对于线性不可分的数据集，我们可以将其映射到高维上，再进行划分。

*C*=1*N*−1∑*Ni*=1*ϕ*(*xi*)*ϕ*(*xi*)*T*=1*N*[*ϕ*(*x*1),...,*ϕ*(*xN*)]⎡⎣⎢⎢⎢*ϕ*(*x*1)*T*...*ϕ*(*xN*)*T*⎤⎦⎥⎥⎥

我们令

*XT*=[*ϕ*(*x*1),...,*ϕ*(*xN*)]

那么

*C*=1*N*−1*XTX*

在这里，*ϕ*(*x*)我们是不知道的，所以上式是没法算的。就算知道了，计算成本也太大了。故引入核函数，我们知道核函数有

*K*=*XXT*=⎡⎣⎢⎢⎢*ϕ*(*x*1)*T*...*ϕ*(*xN*)*T*⎤⎦⎥⎥⎥[*ϕ*(*x*1),⋯,*ϕ*(*xN*)]=⎡⎣⎢⎢⎢*κ*(*x*1,*x*1)⋮*κ*(*xN*,*x*1)...⋱⋯*κ*(*x*1,*xN*)⋮*κ*(*xN*,*xN*)⎤⎦⎥⎥⎥

上述的*K*

我们根据核函数的性质是可以算出来的，现在来看看*K*和*C*之间有没有关系。   
如果要求*K*

的特征值和特征向量的话，我们有下式

(*XXT*)*u*=*λu*

其中，*u*为矩阵*K*的特征向量，*λ*为矩阵*K*的特征值。   
我们对左右两边同时左乘一个*XT*有

*XT*(*XXT*)*u*=*λXTu*

即

(*XTX*)(*XTu*)=*λ*(*XTu*)

又由于(*N*−1)⋅*C*=*XTX*，所以我们发现矩阵*K*和*C*的特征值是相同的，都为*λ*，*C*的特征向量为*XTu*。   
由于我们希望特征向量是单位向量，所以我们对其做一下单位化

*v*=1||*XTu*||*XTu*=1*uTXXTu*‾‾‾‾‾‾‾‾√*XTu*=1*uTKu*‾‾‾‾‾√*XTu*=1*uTλu*‾‾‾‾‾√*XTu*=1*λ*√*XTu*

在上式中，*λ*和*u*可以通过矩阵*K*求得，但是*XT*仍旧是不可知的。那么*C*的特征向量还是算不出来，难道费了这么大的劲，我们白算了？不急，我们接着往下看。虽然求不出*v*，但是*v*并不是我们的最终目标，我们只要知道*x*在*v*上的投影就可以了

*vTϕ*(*xj*)=(1*λ*√*XTu*)*Tϕ*(*xj*)=1*λ*√*uTXϕ*(*xj*)=1*λ*√*uT*⎡⎣⎢⎢⎢*ϕ*(*x*1)*T*⋮*ϕ*(*xN*)*T*⎤⎦⎥⎥⎥*ϕ*(*xj*)=1*λ*√*uT*⎡⎣⎢⎢⎢*κ*(*x*1,*xj*)⋮*κ*(*xN*,*xj*)⎤⎦⎥⎥⎥

上式中所有的量都是可以求得的，也就说我们在没有求出特征向量的情况下，直接算出了样本在特征向量上的投影！   
这样一来问题就解决了！是不是很神奇！

# PCA和KPCA在Python中的使用

在python的sklearn包中，已经对PCA和KPCA进行了实现，我们只需要调用函数即可，非常方便。

## PCA的使用

我们用的数据集是UCI上关于葡萄酒的数据集，得到数据集后对其进行预处理，使得其均值为0。

import pandas as pd

from sklearn.preprocessing import StandardScaler

df = pd.read\_csv('http://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data', header=None)

x, y = df.iloc[:, 1:].values, df.iloc[:, 0].values

sc = StandardScaler()

x = sc.fit\_transform(x)

* 1
* 2
* 3
* 4
* 5
* 6
* 7

这个时候得到的*x*

是一个178×13规模的数据集，也就是说有13个特征，每个特征下有178个数据。   
我们用主成分分析法将13个特征通过线性组合得到一个2

个特征的数据集。

from sklearn.decomposition import PCA

pca = PCA(n\_components=2)

x\_pca = pca.fit\_transform(x)

* 1
* 2
* 3
* 4

然后我们来看下效果

import matplotlib.pyplot as plt

plt.scatter(x\_pca[y==1, 0], x\_pca[y==1, 1], color='red', marker='^', alpha=0.5)

plt.scatter(x\_pca[y==2, 0], x\_pca[y==2, 1], color='blue', marker='o', alpha=0.5)

plt.scatter(x\_pca[y==3, 0], x\_pca[y==3, 1], color='lightgreen', marker='s', alpha=0.5)

plt.xlabel('PC1')

plt.ylabel('PC2')

plt.show()

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8

可以得到

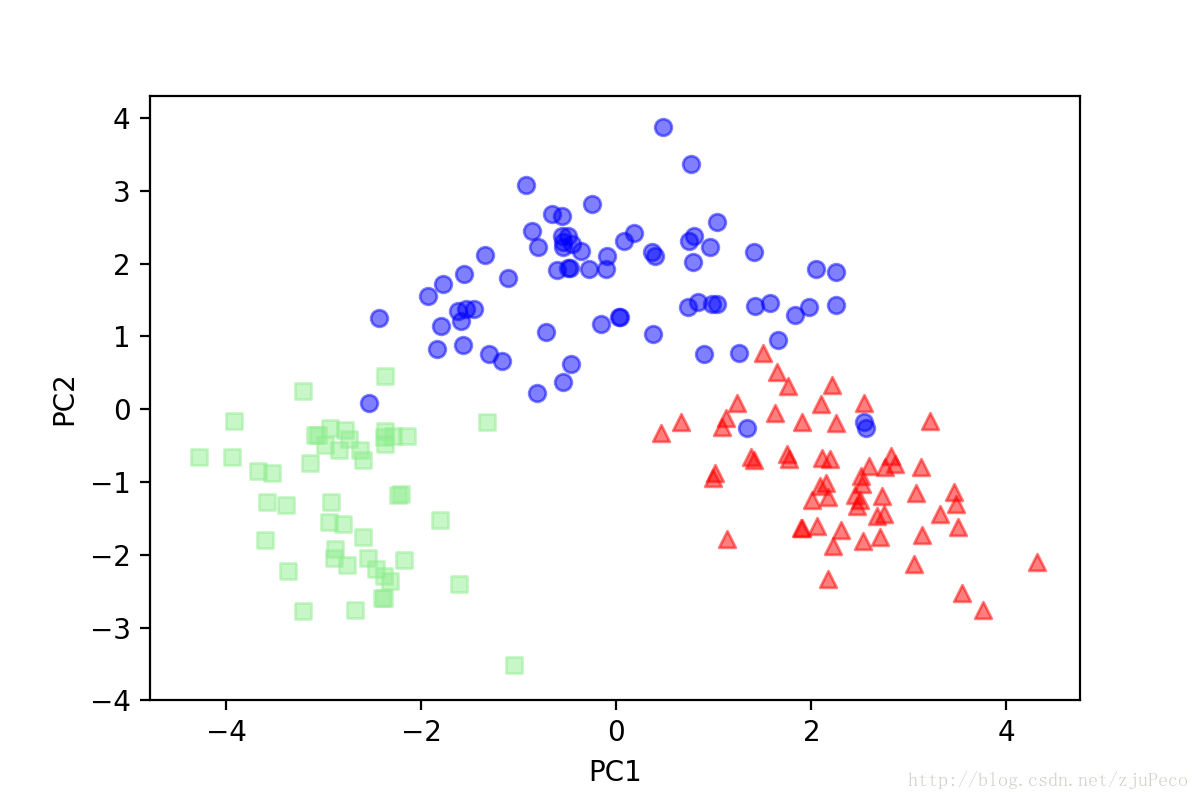


图2：葡萄酒数据主成分分析后效果

很显然，此时已经可以看成是线性可分的数据集了，效果不错。

## KPCA的使用

PCA的使用是有局限性的，如果遇到了，一个像下面这样的线性不可分的数据集，就比较麻烦了。

from sklearn.datasets import make\_moons

x2, y2 = make\_moons(n\_samples=100, random\_state=123)

plt.scatter(x2\_std[y2==0, 0], x2\_std[y2==0, 1], color='red', marker='^', alpha=0.5)

plt.scatter(x2\_std[y2==1, 0], x2\_std[y2==1, 1], color='blue', marker='o', alpha=0.5)

plt.xlabel('PC1')

plt.ylabel('PC2')

plt.show()

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9

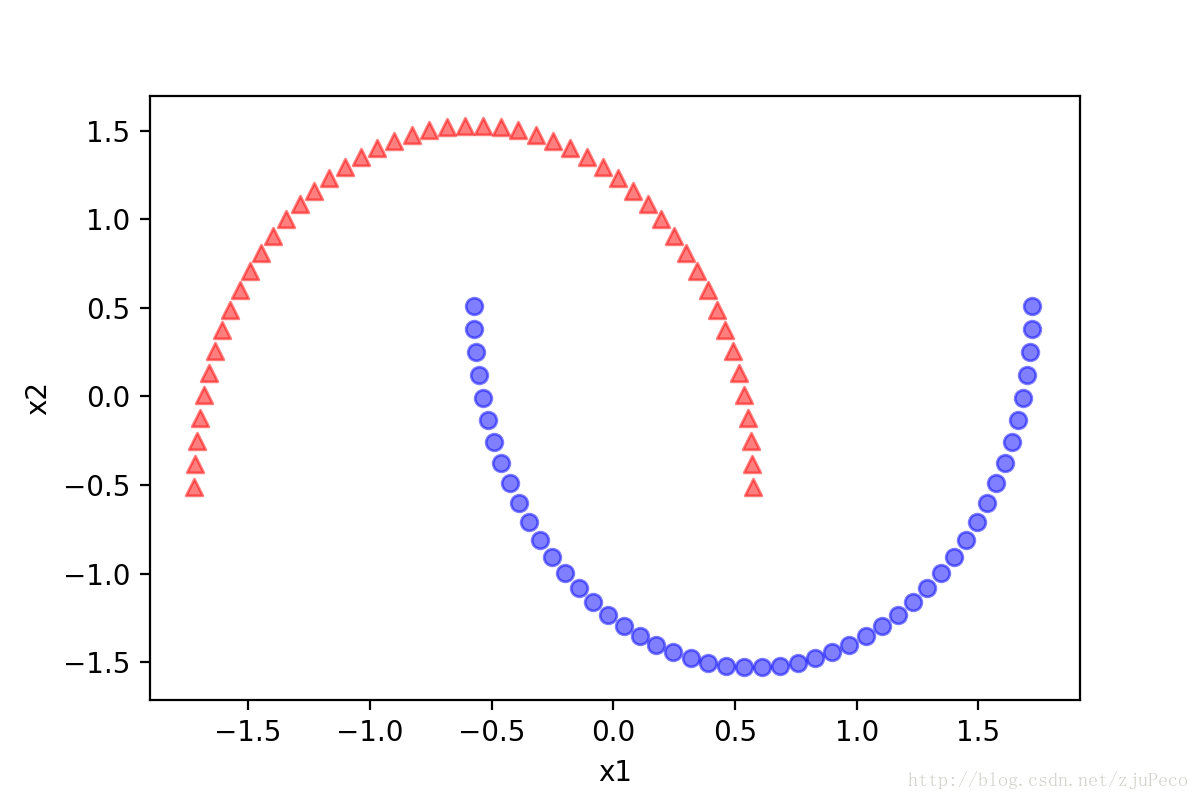


图3：非线性不可分的数据集

不相信的话我们可以用PCA先试下看

x2\_std = sc.fit\_transform(x2)

x\_spca = pca.fit\_transform(x2\_std)

fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(14,6))

ax[0].scatter(x\_spca[y2==0, 0], x\_spca[y2==0, 1], color='red', marker='^', alpha=0.5)

ax[0].scatter(x\_spca[y2==1, 0], x\_spca[y2==1, 1], color='blue', marker='o', alpha=0.5)

ax[1].scatter(x\_spca[y2==0, 0], np.zeros((50,1))+0.02, color='red', marker='^', alpha=0.5)

ax[1].scatter(x\_spca[y2==1, 0], np.zeros((50,1))+0.02, color='blue', marker='o', alpha=0.5)

ax[0].set\_xlabel('PC1')

ax[0].set\_ylabel('PC2')

ax[1].set\_ylim([-1, 1])

ax[1].set\_yticks([])

ax[1].set\_xlabel('PC1')

plt.show()

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14

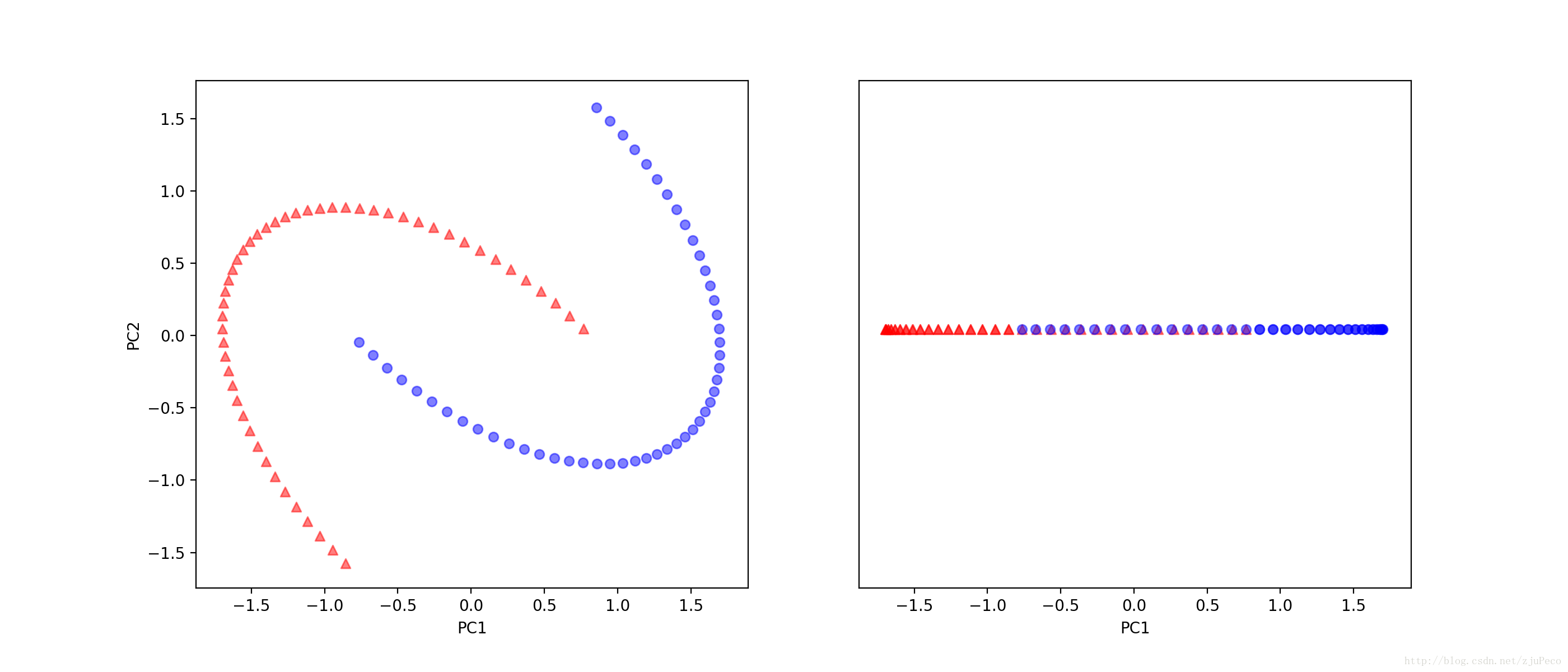


图4：PCA在非线性可分数据集的效果

从图中可以看出，经过主成分分析之后，数据仍旧是线性不可分的。接下来，我们用基于核函数的主成分分析来试下看。

from sklearn.decomposition import KernelPCA

kpca = KernelPCA(n\_components=2, kernel='rbf', gamma=15)

x\_kpca = kpca.fit\_transform(x2\_std)

fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(14,6))

ax[0].scatter(x\_kpca[y2==0, 0], x\_kpca[y2==0, 1], color='red', marker='^', alpha=0.5)

ax[0].scatter(x\_kpca[y2==1, 0], x\_kpca[y2==1, 1], color='blue', marker='o', alpha=0.5)

ax[1].scatter(x\_kpca[y2==0, 0], np.zeros((50,1))+0.02, color='red', marker='^', alpha=0.5)

ax[1].scatter(x\_kpca[y2==1, 0], np.zeros((50,1))+0.02, color='blue', marker='o', alpha=0.5)

ax[0].set\_xlabel('PC1')

ax[0].set\_ylabel('PC2')

ax[1].set\_ylim([-1, 1])

ax[1].set\_yticks([])

ax[1].set\_xlabel('PC1')

plt.show()

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16

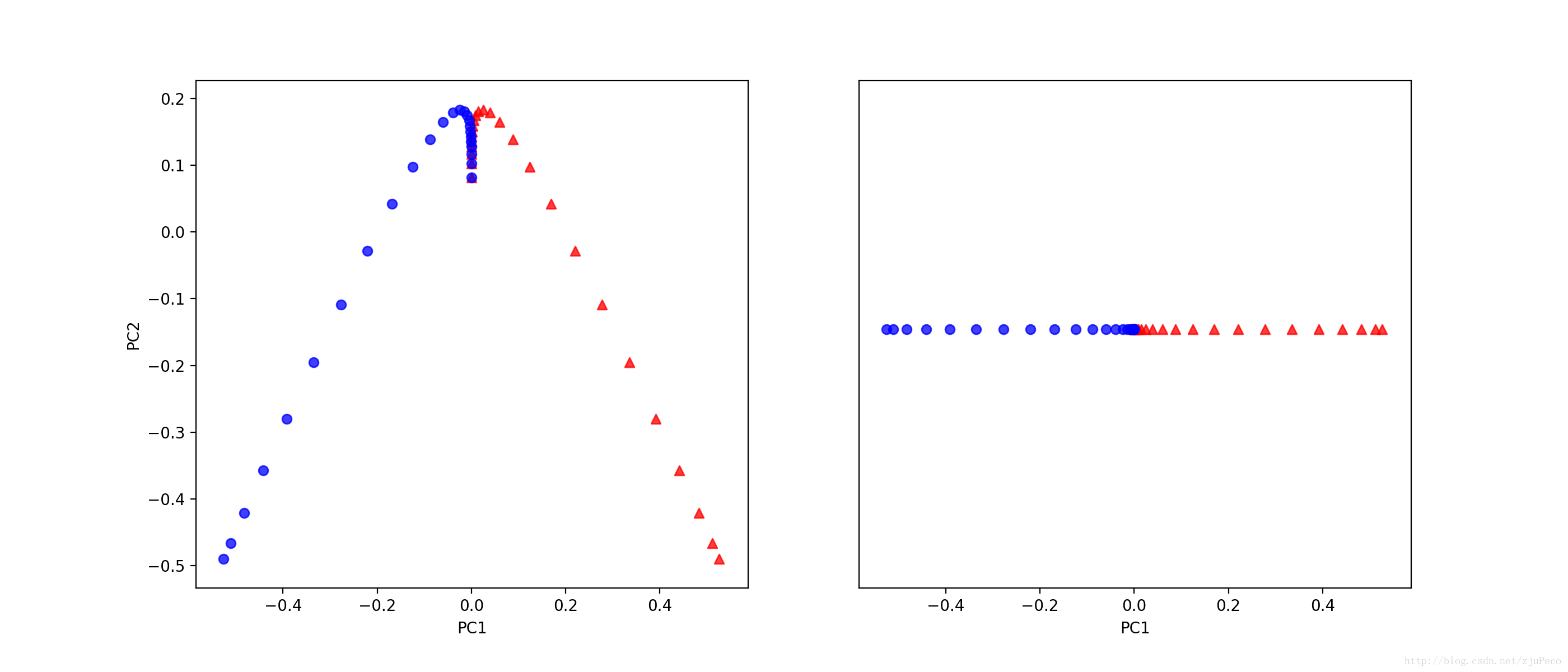


图5：KPCA在非线性可分数据集的效果

由图可知，只需要把数据集投影到经变换后的特征*PC*1上就可以实现线性划分了，这个时候只需要一个特征*PC*1就够了。

# 参考文献

[1] <https://www.youtube.com/watch?v=G2NRnh7W4NQ&t=1s>   
[2] Raschka S. Python Machine Learning[M]. Packt Publishing, 2015.