菜菜的scikit-learn课堂04

sklearn中的降维算法PCA和SVD

小伙伴们晚上好~o(￣▽￣)ブ

我是菜菜，这里是我的sklearn课堂第四期，今晚的直播内容是降维算法PCA和SVD~

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 0.20.0 （你的版本至少要0.19

Numpy 1.15.3, **Pandas** 0.23.4, **Matplotlib** 3.0.1, **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04

sklearn中的降维算法PCA和SVD

1 概述

1.1 从什么叫"维度"说开来

在过去的三周里，我们已经带大家认识了两个算法和数据预处理过程。期间，我们不断提到一些语言，比如说：随机森林是通过随机抽取特征来建树，以避免高维计算；再比如说，sklearn中导入特征矩阵，必须是至少二维；上周我们讲解特征工程，还特地提到了，特征选择的目的是通过降维来降低算法的计算成本……这些语言都很正常地被我用来使用，直到有一天，一个小伙伴问了我，"维度"到底是什么？

对于**数组和Series**来说，**维度就是功能shape返回的结果，shape中返回了几个数字，就是几维。**索引以外的数据，不分行列的叫一维（此时shape返回唯一的维度上的数据个数），有行列之分叫二维（shape返回行x列），也称为表。一张表最多二维，复数的表构成了更高的维度。当一个数组中存在2张3行4列的表时，shape返回的是(更高维，行，列)。当数组中存在2组2张3行4列的表时，数据就是4维，shape返回(2,2,3,4)。

	一维	二维	三维

数组
Series
	`array([0., 0.])`
	shape (2,)
		`array([[0., 0., 0., 0.],`
		`[0., 0., 0., 0.],`
		`[0., 0., 0., 0.]])`
		shape (3,4)
			`array([[[0., 0., 0., 0.],`
			`[0., 0., 0., 0.],`
			`[0., 0., 0., 0.]],`
			
			`[[0., 0., 0., 0.],`
			`[0., 0., 0., 0.],`
			`[0., 0., 0., 0.]]])`
			shape (2,3,4)

数组中的每一张表，都可以是一个**特征矩阵**或一个DataFrame，这些结构永远只有一张表，所以一定有行列，其中行是样本，列是特征。针对每一张表，**维度指的是样本的数量或特征的数量，一般无特别说明，指的都是特征的数量。**除了索引之外，一个特征是一维，两个特征是二维，n个特征是n维。

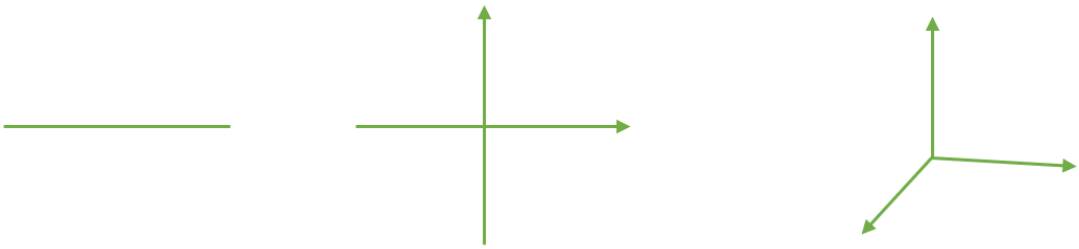
		特征1
DataFrame 特征矩阵	0	0.0
	1	0.0

	特征1	特征2
0	0.0	0.0
1	0.0	0.0

	特征1	特征2	特征3
0	0.0	0.0	0.0
1	0.0	0.0	0.0

对图像来说，维度就是图像中特征向量的数量。特征向量可以理解为是坐标轴，一个特征向量定义一条直线，是一维，两个相互垂直的特征向量定义一个平面，即一个直角坐标系，就是二维，三个相互垂直的特征向量定义一个空间，即一个立体直角坐标系，就是三维。三个以上的特征向量相互垂直，定义人眼无法看见，也无法想象的高维空间。

图像



降维算法中的"降维"，指的是降低特征矩阵中特征的数量。上周的课中我们说过，降维的目的是为了让**算法运算更快，效果更好，**但其实还有另一种需求：**数据可视化。**从上面的图我们其实可以看得出，图像和特征矩阵的维度是可以相互对应的，即一个特征对应一个特征向量，对应一条坐标轴。所以，三维及以下的特征矩阵，是可以被可视化的，这可以帮助我们很快地理解数据的分布，而三维以上特征矩阵的则不能被可视化，数据的性质也就比较难理解。

1.2 sklearn中的降维算法

sklearn中降维算法都被包括在模块decomposition中，这个模块本质是一个矩阵分解模块。在过去的十年中，如果要讨论算法进步的先锋，矩阵分解可以说是独树一帜。矩阵分解可以用在降维，深度学习，聚类分析，数据预处理，低纬度特征学习，推荐系统，大数据分析等领域。在2006年，Netflix曾经举办了一个奖金为100万美元的推荐系统算法比赛，最后的获奖者就使用了矩阵分解中的明星：奇异值分解SVD。(～o￣3￣)～菊安酱会讲SVD在推荐系统中的应用，大家不要错过！

类	说明
主成分分析	
decomposition.PCA	主成分分析（PCA）
decomposition.IncrementalPCA	增量主成分分析（IPCA）
decomposition.KernelPCA	核主成分分析（KPCA）
decomposition.MiniBatchSparsePCA	小批量稀疏主成分分析
decomposition.SparsePCA	稀疏主成分分析（SparsePCA）
decomposition.TruncatedSVD	截断的SVD（aka LSA）
因子分析	
decomposition.FactorAnalysis	因子分析（FA）
独立成分分析	
decomposition.FastICA	独立成分分析的快速算法
字典学习	
decomposition.DictionaryLearning	字典学习
decomposition.MiniBatchDictionaryLearning	小批量字典学习
decomposition.dict_learning	字典学习用于矩阵分解
decomposition.dict_learning_online	在线字典学习用于矩阵分解
高级矩阵分解	
decomposition.LatentDirichletAllocation	具有在线变分贝叶斯算法的隐含狄利克雷分布
decomposition.NMF	非负矩阵分解（NMF）
其他矩阵分解	
decomposition.SparseCoder	稀疏编码

SVD和主成分分析PCA都属于矩阵分解算法中的入门算法，都是通过分解特征矩阵来进行降维，它们也是我们今天要讲解的重点。虽然是入门算法，却不代表PCA和SVD简单：下面两张图是我在一篇SVD的论文中随意截取的两页，可以看到满满的数学公式（基本是线性代数）。要想在短短的一个小时内，给大家讲明白这些公式，我讲完不吐血大家听完也吐血了。所以今天，我会用最简单的方式为大家呈现降维算法的原理，但这注定意味着大家无法看到这个算法的全貌，**在机器学习中逃避数学是邪道**，所以更多原理大家自己去阅读。

6.2. Matrices for which matrix–vector products can be rapidly evaluated. In many problems in data mining and scientific computing, the cost T_{mult} of performing the matrix–vector multiplication $\boldsymbol{x} \mapsto \boldsymbol{A}\boldsymbol{x}$ is substantially smaller than the nominal cost $\mathrm{O}(mn)$ for the dense case. It is not uncommon that $\mathrm{O}(m+n)$ flops suffice. Standard examples include (i) very sparse matrices; (ii) structured matrices, such as Tōplitz operators, that can be applied using the FFT or other means; and (iii) matrices that arise from physical problems, such as discretized integral operators, that can be applied via, e.g., the fast multipole method [66].

Suppose that both $\boldsymbol{A}$ and $\boldsymbol{A}^*$ admit fast multiplies. The appropriate randomized approach for this scenario completes Stage A using Algorithm 4.1 with p constant (for the fixed-rank problem) or Algorithm 4.2 (for the fixed-precision problem) at a cost of $(k+p)\,T_{\text{mult}} + \mathrm{O}(k^2 m)$ flops. For Stage B, we invoke Algorithm 5.1, which requires $(k+p)\,T_{\text{mult}} + \mathrm{O}(k^2(m+n))$ flops. The total cost T_{sparse} satisfies

$$T_{\text{sparse}} = 2\,(k+p)\,T_{\text{mult}} + \mathrm{O}(k^2(m+n)). \tag{6.2}$$

As a rule of thumb, the approximation error of this procedure satisfies

$$\|\boldsymbol{A} - \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^*\| \lesssim \sqrt{kn}\cdot\sigma_{k+1}. \tag{6.3}$$

The estimate (6.3) follows from Corollary 10.9 and the discussion in §5.1. Actual errors are usually smaller.

When the singular spectrum of $\boldsymbol{A}$ decays slowly, we can incorporate q iterations of the power method (Algorithm 4.3) to obtain superior solutions to the fixed-rank problem. The computational cost increases to, cf. (6.2),

$$T_{\text{sparse}} = (2q+2)\,(k+p)\,T_{\text{mult}} + \mathrm{O}(k^2(m+n)), \tag{6.4}$$

while the error (6.3) improves to

$$\|\boldsymbol{A} - \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^*\| \lesssim (kn)^{1/2(2q+1)}\cdot\sigma_{k+1}. \tag{6.5}$$

The estimate (6.5) takes into account the discussion in §10.4. The power scheme can also be adapted for the fixed-precision problem (§4.5).

In this setting, the classical prescription for obtaining a partial SVD is some variation of a Krylov-subspace method; see §3.3.4. These methods exhibit great diversity, so it is hard to specify a "typical" computational cost. To a first approximation, it is fair to say that in order to obtain an approximate SVD of rank k, the cost of a numerically stable implementation of a Krylov method is no less than the cost (6.2) with p set to zero. At this price, the Krylov method often obtains better accuracy than the basic randomized method obtained by combining Algorithms 4.1 and 5.1, especially for matrices whose singular values decay slowly. On the other hand, the randomized schemes are inherently more robust and allow much more freedom in organizing the computation to suit a particular application or a particular hardware architecture. The latter point is in practice of crucial importance because it is usually much faster to apply a matrix to k vectors simultaneously than it is to execute k matrix–vector multiplications consecutively. In practice, blocking and parallelism can lead to enough gain that a few steps of the power method (Algorithm 4.3) can be performed more quickly than k steps of a Krylov method.

REMARK 6.2. Any comparison between randomized sampling schemes and Krylov variants becomes complicated because of the fact that "basic" Krylov schemes such

PROPOSITION 8.2 (Perturbation of Inverses). *Suppose that* $\boldsymbol{M} \succcurlyeq \boldsymbol{0}$. *Then*

$$\mathbf{I} - (\mathbf{I}+\boldsymbol{M})^{-1} \preccurlyeq \boldsymbol{M}$$

Proof. Define $\boldsymbol{R} = \boldsymbol{M}^{1/2}$, the psd square root of $\boldsymbol{M}$ promised by [72, Thm. 7.2.6]. We have the chain of relations

$$\mathbf{I} - (\mathbf{I}+\boldsymbol{R}^2)^{-1} = (\mathbf{I}+\boldsymbol{R}^2)^{-1}\boldsymbol{R}^2 = \boldsymbol{R}(\mathbf{I}+\boldsymbol{R}^2)^{-1}\boldsymbol{R} \preccurlyeq \boldsymbol{R}^2.$$

The first equality can be verified algebraically. The second holds because rational functions of a diagonalizable matrix, such as $\boldsymbol{R}$, commute. The last relation follows from the conjugation rule because $(\mathbf{I}+\boldsymbol{R}^2)^{-1} \preccurlyeq \mathbf{I}$. ∎

Next, we present a generalization of the fact that the spectral norm of a psd matrix is controlled by its trace.

PROPOSITION 8.3. *We have* $\|\boldsymbol{M}\| \le \|\boldsymbol{A}\| + \|\boldsymbol{C}\|$ *for each partitioned psd matrix*

$$\boldsymbol{M} = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{B}^* & \boldsymbol{C} \end{bmatrix}.$$

Proof. The variational characterization (8.1) of the spectral norm implies that

$$\begin{aligned} \|\boldsymbol{M}\| &= \sup_{\|\boldsymbol{x}\|^2+\|\boldsymbol{y}\|^2=1} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{bmatrix}^* \begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{B}^* & \boldsymbol{C} \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{bmatrix} \\ &\le \sup_{\|\boldsymbol{x}\|^2+\|\boldsymbol{y}\|^2=1} \left(\|\boldsymbol{A}\|\,\|\boldsymbol{x}\|^2 + 2\,\|\boldsymbol{B}\|\,\|\boldsymbol{x}\|\,\|\boldsymbol{y}\| + \|\boldsymbol{C}\|\,\|\boldsymbol{y}\|^2 \right). \end{aligned}$$

The block generalization of Hadamard's psd criterion [72, Thm. 7.7.7] states that $\|\boldsymbol{B}\|^2 \le \|\boldsymbol{A}\|\,\|\boldsymbol{C}\|$. Thus,

$$\|\boldsymbol{M}\| \le \sup_{\|\boldsymbol{x}\|^2+\|\boldsymbol{y}\|^2=1} \left(\|\boldsymbol{A}\|^{1/2}\|\boldsymbol{x}\| + \|\boldsymbol{C}\|^{1/2}\|\boldsymbol{y}\| \right)^2 = \|\boldsymbol{A}\| + \|\boldsymbol{C}\|.$$

This point completes the argument. ∎

8.2. Orthogonal projectors. An *orthogonal projector* is an Hermitian matrix $\boldsymbol{P}$ that satisfies the polynomial $\boldsymbol{P}^2 = \boldsymbol{P}$. This identity implies $\boldsymbol{0} \preccurlyeq \boldsymbol{P} \preccurlyeq \mathbf{I}$. An orthogonal projector is completely determined by its range. For a given matrix $\boldsymbol{M}$, we write $\boldsymbol{P}_{\boldsymbol{M}}$ for the unique orthogonal projector with $\text{range}(\boldsymbol{P}_{\boldsymbol{M}}) = \text{range}(\boldsymbol{M})$. When $\boldsymbol{M}$ has full column rank, we can express this projector explicitly:

$$\boldsymbol{P}_{\boldsymbol{M}} = \boldsymbol{M}(\boldsymbol{M}^*\boldsymbol{M})^{-1}\boldsymbol{M}^*. \tag{8.3}$$

The orthogonal projector onto the complementary subspace, $\text{range}(\boldsymbol{P})^{\perp}$, is the matrix $\mathbf{I} - \boldsymbol{P}$. Our argument hinges on several other facts about orthogonal projectors.

PROPOSITION 8.4. *Suppose* $\boldsymbol{U}$ *is unitary. Then* $\boldsymbol{U}^*\boldsymbol{P}_{\boldsymbol{M}}\boldsymbol{U} = \boldsymbol{P}_{\boldsymbol{U}^*\boldsymbol{M}}$.

Proof. Abbreviate $\boldsymbol{P} = \boldsymbol{U}^*\boldsymbol{P}_{\boldsymbol{M}}\boldsymbol{U}$. It is clear that $\boldsymbol{P}$ is an orthogonal projector since it is Hermitian and $\boldsymbol{P}^2 = \boldsymbol{P}$. Evidently,

$$\text{range}(\boldsymbol{P}) = \boldsymbol{U}^*\,\text{range}(\boldsymbol{M}) = \text{range}(\boldsymbol{U}^*\boldsymbol{M}).$$

2 PCA与SVD

在降维过程中，我们会减少特征的数量，这意味着删除数据，数据量变少则表示模型可以获取的信息会变少，模型的表现可能会因此受影响。同时，在高维数据中，必然有一些特征是不带有有效的信息的（比如噪音），或者有一些特征带有的信息和其他一些特征是重复的（比如一些特征可能会线性相关）。我们希望能够找出一种办法来帮助我们衡量特征上所带的信息量，让我们在降维的过程中，能够**即减少特征的数量，又保留大部分有效信息**——将那些带有重复信息的特征合并，并删除那些带无效信息的特征等等——逐渐创造出能够代表原特征矩阵大部分信息的，特征更少的，新特征矩阵。

上周的特征工程课中，我们提到过一种重要的特征选择方法：方差过滤。如果一个特征的方差很小，则意味着这个特征上很可能有大量取值都相同（比如90%都是1，只有10%是0，甚至100%是1），那这一个特征的取值对样本而言就没有区分度，这种特征就不带有有效信息。从方差的这种应用就可以推断出，如果一个特征的方差很大，则说明这个特征上带有大量的信息。因此，在降维中，**PCA使用的信息量衡量指标，就是样本方差，又称可解释性方差，方差越大，特征所带的信息量越多。**

$$Var = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \hat{x})^2$$

Var代表一个特征的方差，n代表样本量，xi代表一个特征中的每个样本取值，xhat代表这一列样本的均值。

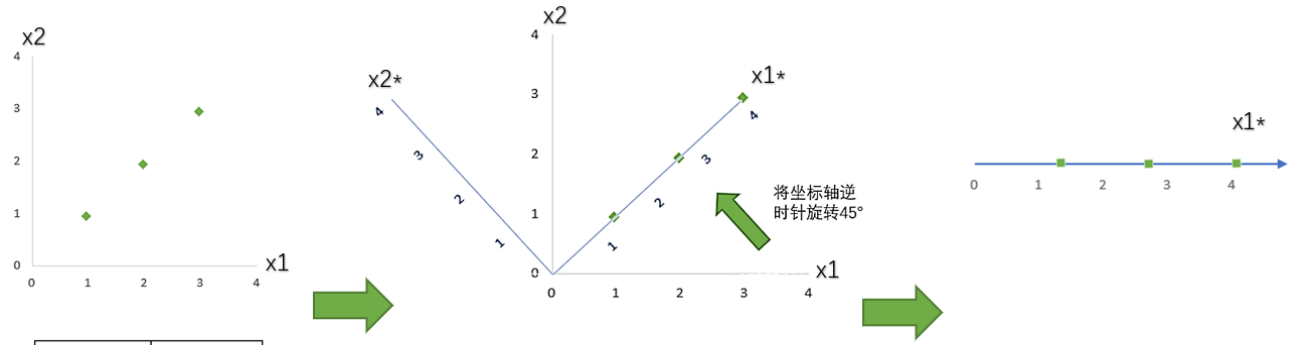
面试高危问题

方差计算公式中为什么除数是n-1?
这是为了得到样本方差的无偏估计，更多大家可以自己去探索~

2.1 降维究竟是怎样实现?

class `sklearn.decomposition.PCA` (*n_components=None*, *copy=True*, *whiten=False*, *svd_solver='auto'*, *tol=0.0*, *iterated_power='auto'*, *random_state=None*)

PCA作为矩阵分解算法的核心算法，其实没有太多参数，但不幸的是每个参数的意义和运用都很难，因为几乎每个参数都涉及到高深的数学原理。为了参数的运用和意义变得明朗，我们来看一组简单的二维数据的降维。



x1	x2
1	1
2	2
3	3

x1*	x2*
$\sqrt{2}$	0
$2\sqrt{2}$	0
$3\sqrt{2}$	0

x1*
$\sqrt{2}$
$2\sqrt{2}$
$3\sqrt{2}$

我们现在有一组简单的数据，有特征x1和x2，三个样本数据的坐标点分别为(1,1)，(2,2)，(3,3)。我们可以让x1和x2分别作为两个特征向量，很轻松地用一个二维平面来描述这组数据。这组数据现在每个特征的均值都为2，方差则等于：

$$x1\_var = x2\_var = \frac{(1-2)^2 + (2-2)^2 + (3-2)^2}{2} = 1$$

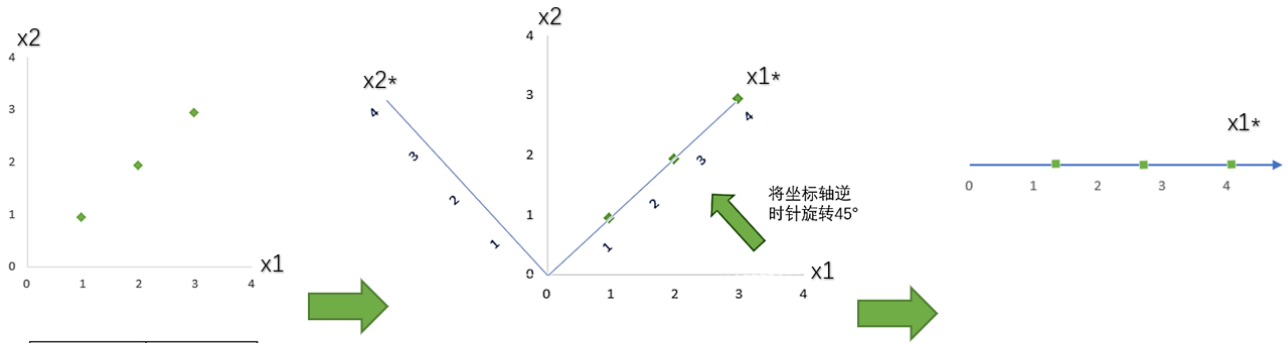
每个特征的数据一模一样，因此方差也都为1，数据的方差总和是2。

现在我们的目标是：只用一个特征向量来描述这组数据，即将二维数据降为一维数据，并且尽可能地保留信息量，即让数据的总方差尽量靠近2。于是，我们将原本的直角坐标系逆时针旋转45°，形成了新的特征向量x1*和x2*组成的新平面，在这个新平面中，三个样本数据的坐标点可以表示为$(\sqrt{2}, 0)$，$(2\sqrt{2}, 0)$，$(3\sqrt{2}, 0)$。可以注意到，x2*上的数值此时都变成了0，因此x2*明显不带有任何有效信息了（此时x2*的方差也为0了）。此时，x1*特征上的数据均值是$2\sqrt{2}$，而方差则可表示成：

$$x2^*\_var = \frac{(\sqrt{2} - 2\sqrt{2})^2 + (2\sqrt{2} - 2\sqrt{2})^2 + (3\sqrt{2} - 2\sqrt{2})^2}{2} = 2$$

x1*上的数据均值为0，方差也为0。

此时，我们根据信息含量的排序，取信息含量最大的一个特征，因为我们想要的是一维数据。所以我们可以将x2*删除，同时也删除图中的x2*特征向量，剩下的x1*就代表了曾经需要两个特征来代表的三个样本点。通过旋转原有特征向量组成的坐标轴来找到新特征向量和新坐标平面，我们将三个样本点的信息压缩到了一条直线上，实现了二维变一维，并且尽量保留原始数据的信息。一个成功的降维，就实现了。



x1	x2
1	1
2	2
3	3

x1*	x2*
$\sqrt{2}$	0
$2\sqrt{2}$	0
$3\sqrt{2}$	0

x1*
$\sqrt{2}$
$2\sqrt{2}$
$3\sqrt{2}$

不难注意到，在这个降维过程中，有几个重要的步骤：

过程	二维特征矩阵	n维特征矩阵
1	输入原数据，结构为 (3,2) 找出原本的2个特征对应的直角坐标系，本质是找出这2个特征构成的2维平面	输入原数据，结构为 (m,n) 找出原本的n个特征向量构成的n维空间V
2	决定降维后的特征数量：1	决定降维后的特征数量：k
3	旋转，找出一个新坐标系 本质是找出2个新的特征向量，以及它们构成的新2维平面 新特征向量让数据能够被压缩到少数特征上，并且总信息量不损失太多	通过某种变化，找出n个新的特征向量，以及它们构成的新n维空间V
4	找出数据点在新坐标系上，2个新坐标轴上的坐标	找出原始数据在新特征空间V中的n个新特征向量上对应的值，即“将数据映射到新空间中”
5	选取第1个方差最大的特征向量，删掉没有被选中的特征，成功将2维平面降为1维	选取前k个信息量最大的特征，删掉没有被选中的特征，成功将n维空间V降为k维

在步骤3当中，我们用来**找出n个新特征向量，让数据能够被压缩到少数特征上并且总信息量不损失太多的技术就是矩阵分解。**PCA和SVD是两种不同的降维算法，但他们都遵从上面的过程来实现降维，只是两种算法中矩阵分解的方法不同，信息量的衡量指标不同罢了。PCA使用方差作为信息量的衡量指标，并且特征值分解来找出空间V。降维时，它会通过一系列数学的神秘操作（比如说，产生协方差矩阵$\frac{1}{n}XX^T$）将特征矩阵X分解为以下三个矩阵，其中Q和Q^{-1}是辅助的矩阵，Σ是一个对角矩阵（即除了对角线上有值，其他位置都是0的矩阵），其对角线上的元素就是方差。降维完成之后，PCA找到的每个新特征向量就叫做“主成分”，而被丢弃的特征向量被认为信息量很少，这些信息很可能就是噪音。

$$X \rightarrow 数学神秘的宇宙 \rightarrow Q\Sigma Q^{-1}$$

而SVD使用奇异值分解来找出空间V，其中Σ也是一个对角矩阵，不过它对角线上的元素是奇异值，这也是SVD中用来衡量特征上的信息量的指标。U和V^{T}分别是左奇异矩阵和右奇异矩阵，也都是辅助矩阵。

$$X \rightarrow 另一个数学神秘的宇宙 \rightarrow U\Sigma V^{T}$$

在数学原理中，无论是PCA和SVD都需要遍历所有的特征和样本来计算信息量指标。并且在矩阵分解的过程之中，会产生比原来的特征矩阵更大的矩阵，比如原数据的结构是(m,n)，在矩阵分解中为了找出最佳新特征空间V，可能需要产生(n,n)，(m,m)大小的矩阵，还需要产生协方差矩阵去计算更多的信息。而现在无论是Python还是R，或者其他的任何语言，在大型矩阵运算上都不是特别擅长，无论代码如何简化，我们不可避免地要等待计算机去完成这个非常庞大的数学计算过程。因此，降维算法的计算量很大，运行比较缓慢，但无论如何，它们的功能无可替代，它们依然是机器学习领域的宠儿。

思考：PCA和特征选择技术都是特征工程的一部分，它们有什么不同?

特征工程中有三种方式：特征提取，特征创造和特征选择。仔细观察上面的降维例子和上周我们讲解过的特征选择，你发现有什么不同了吗?

特征选择是从已存在的特征中选取携带信息最多的，选完之后的特征依然具有可解释性，我们依然知道这个特征在原数据的哪个位置，代表着原数据上的什么含义。

而PCA，是将已存在的特征进行压缩，降维完毕后的特征不是原本的特征矩阵中的任何一个特征，而是通过某些方式组合起来的新特征。通常来说，**在新的特征矩阵生成之前，我们无法知晓PCA都建立了怎样的新特征向量，新特征矩阵生成之后也不具有可读性**，我们无法判断新特征矩阵的特征是从原数据中的什么特征组合而来，新特征虽然带有原始数据的信息，却已经不是原数据上代表着的含义了。以PCA为代表的降维算法因此是特征创造（feature creation，或feature construction）的一种。

可以想见，PCA一般不适用于探索特征和标签之间的关系的模型（如线性回归），因为无法解释的新特征和标签之间的关系不具有意义。在线性回归模型中，我们使用特征选择。

2.2 重要参数n_components

n_components是我们降维后需要的维度，即降维后需要保留的特征数量，降维流程中第二步里需要确认的k值，一般输入[0, min(X.shape)]范围中的整数。一说到K，大家可能都会想到，类似于KNN中的K和随机森林中的n_estimators，这是一个需要我们人为去确认的超参数，并且我们设定的数字会影响到模型的表现。如果留下的特征太多，就达不到降维的效果，如果留下的特征太少，那新特征向量可能无法容纳原始数据集中的大部分信息，因此，n_components既不能太大也不能太小。那怎么办呢？

可以先从我们的降维目标说起：如果我们希望可视化一组数据来观察数据分布，我们往往将数据降到三维以下，很多时候是二维，即n_components的取值为2。

2.2.1 迷你案例：高维数据的可视化

1. **调用库和模块**

```
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.decomposition import PCA
```

2. **提取数据集**

```
iris = load_iris()
y = iris.target
X = iris.data
#作为数组，X是几维？
X.shape
#作为数据表或特征矩阵，X是几维？
import pandas as pd
pd.DataFrame(X)
```

3. **建模**

```
#调用PCA
pca = PCA(n_components=2)               #实例化
pca = pca.fit(X)                        #拟合模型
X_dr = pca.transform(X)                 #获取新矩阵

X_dr
#也可以fit_transform一步到位
#X_dr = PCA(2).fit_transform(X)
```

4. **可视化**

```
#要将三种鸢尾花的数据分布显示在二维平面坐标系中，对应的两个坐标（两个特征向量）应该是三种鸢尾花降维后的x1和x2，怎样才能取出三种鸢尾花下不同的x1和x2呢？

X_dr[y == 0, 0] #这里是布尔索引，看出来了么？

#要展示三中分类的分布，需要对三种鸢尾花分别绘图
#可以写成三行代码，也可以写成for循环
```

```
"""
plt.figure()
plt.scatter(X_dr[y==0, 0], X_dr[y==0, 1], c="red", label=iris.target_names[0])
plt.scatter(X_dr[y==1, 0], X_dr[y==1, 1], c="black", label=iris.target_names[1])
plt.scatter(X_dr[y==2, 0], X_dr[y==2, 1], c="orange", label=iris.target_names[2])
plt.legend()
plt.title('PCA of IRIS dataset')
plt.show()
"""

colors = ['red', 'black', 'orange']
iris.target_names

plt.figure()
for i in [0, 1, 2]:
    plt.scatter(X_dr[y == i, 0]
                ,X_dr[y == i, 1]
                ,alpha=.7
                ,c=colors[i]
                ,label=iris.target_names[i]
               )
plt.legend()
plt.title('PCA of IRIS dataset')
plt.show()
```

鸢尾花的分布被展现在我们眼前了，明显这是一个分簇的分布，并且每个簇之间的分布相对比较明显，也许versicolor和virginia这两种花之间会有一些分类错误，但setosa肯定不会被分错。这样的数据很容易分类，可以遇见，KNN，随机森林，神经网络，朴素贝叶斯，Adaboost这些分类器在鸢尾花数据集上，未调整的时候都可以有95%上下的准确率。

6. **探索降维后的数据**

```
#属性explained_variance_，查看降维后每个新特征向量上所带的信息量大小（可解释性方差的大小）
pca.explained_variance_

#属性explained_variance_ratio，查看降维后每个新特征向量所占的信息量占原始数据总信息量的百分比
#又叫做可解释方差贡献率
pca.explained_variance_ratio_
#大部分信息都被有效地集中在了第一个特征上

pca.explained_variance_ratio_.sum()
```

7. **选择最好的n_components：累积可解释方差贡献率曲线**

当参数n_components中不填写任何值，则默认返回min(X.shape)个特征，一般来说，样本量都会大于特征数目，所以什么都不填就相当于转换了新特征空间，但没有减少特征的个数。一般来说，不会使用这种输入方式。但我们却可以使用这种输入方式来画出累计可解释方差贡献率曲线，以此选择最好的n_components的整数取值。

累积可解释方差贡献率曲线是一条以降维后保留的特征个数为横坐标，降维后新特征矩阵捕捉到的可解释方差贡献率为纵坐标的曲线，能够帮助我们决定n_components最好的取值。

```
import numpy as np
pca_line = PCA().fit(X)
plt.plot([1,2,3,4],np.cumsum(pca_line.explained_variance_ratio_))
plt.xticks([1,2,3,4]) #这是为了限制坐标轴显示为整数
plt.xlabel("number of components after dimension reduction")
plt.ylabel("cumulative explained variance ratio")
plt.show()
```

2.2.2 最大似然估计自选超参数

除了输入整数，n_components还有哪些选择呢？之前我们提到过，矩阵分解的理论发展在业界独树一帜，勤奋智慧的数学大神Minka, T.P.在麻省理工学院媒体实验室做研究时找出了让PCA用最大似然估计(maximum likelihood estimation)自选超参数的方法，输入“mle”作为n_components的参数输入，就可以调用这种方法。

```
pca_mle = PCA(n_components="mle")
pca_mle = pca_mle.fit(X)
X_mle = pca_mle.transform(X)

X_mle
#可以发现，mle为我们自动选择了3个特征

pca_mle.explained_variance_ratio_.sum()
#得到了比设定2个特征时更高的信息含量，对于鸢尾花这个很小的数据集来说，3个特征对应这么高的信息含量，并不需要去纠结于只保留2个特征，毕竟三个特征也可以可视化
```

2.2.3 按信息量占比选超参数

输入[0,1]之间的浮点数，并且让参数svd_solver =='full'，表示希望降维后的总解释性方差占比大于n_components指定的百分比，即是说，希望保留百分之多少的信息量。比如说，如果我们希望保留97%的信息量，就可以输入n_components = 0.97，PCA会自动选出能够让保留的信息量超过97%的特征数量。

```
pca_f = PCA(n_components=0.97,svd_solver="full")
pca_f = pca_f.fit(X)
X_f = pca_f.transform(X)

pca_f.explained_variance_ratio_
```

2.3 PCA中的SVD

2.3.1 PCA中的SVD哪里来?

细心的小伙伴可能注意到了，svd_solver是奇异值分解器的意思，为什么PCA算法下面会有有关奇异值分解的参数？不是两种算法么？我们之前曾经提到过，PCA和SVD涉及了大量的矩阵计算，两者都是运算量很大的模型，但其实，SVD有一种惊人的数学性质，即是**它可以跳过数学神秘的宇宙，不计算协方差矩阵，直接找出一个新特征向量组成的n维空间**，而这个n维空间就是奇异值分解后的右矩阵V^T（所以一开始在讲解降维过程时，我们说"生成新特征向量组成的空间V"，并非巧合，而是特指奇异值分解中的矩阵V^T）。

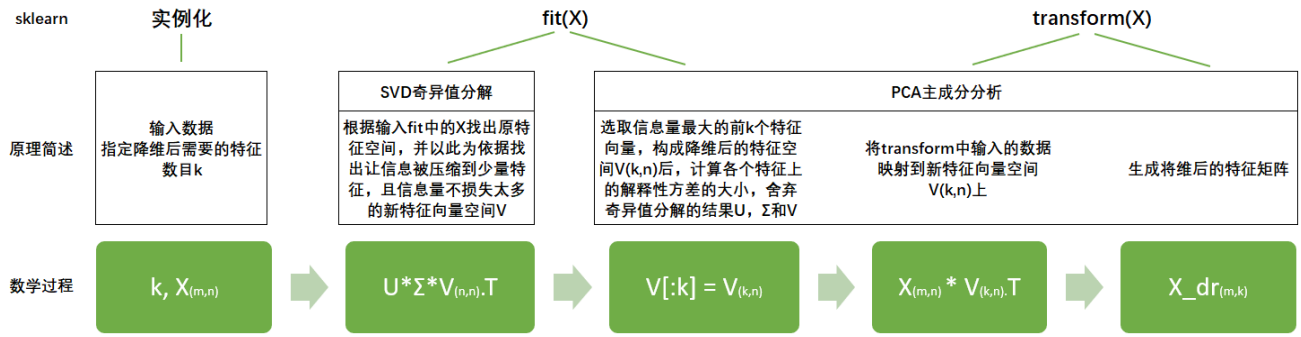
$$传统印象中的SVD：\ X \rightarrow 数学神秘的宇宙 \rightarrow U\Sigma V^T$$
$$其实会开挂的SVD：\ X \rightarrow 一个比起PCA简化非常多的数学过程 \rightarrow V^T$$

右奇异矩阵V^T有着如下性质：

$$X_{dr} = X * V[:k]^T$$

k就是n_components，是我们降维后希望得到的维度。若X为(m,n)的特征矩阵，V^T就是结构为(n,n)的矩阵，取这个矩阵的前k行（进行切片），即将V转换为结构为(k,n)的矩阵。而$V_{(k,n)}^T$与原特征矩阵X相乘，即可得到降维后的特征矩阵X_dr。**这是说，奇异值分解可以不计算协方差矩阵等等结构复杂计算冗长的矩阵，就直接求出新特征空间和降维后的特征矩阵。**

简而言之，SVD在矩阵分解中的过程比PCA简单快速，虽然两个算法都走一样的分解流程，但SVD可以作弊耍赖直接算出V。但是遗憾的是，SVD的信息量衡量指标比较复杂，要理解"奇异值"远不如理解"方差"来得容易，因此，sklearn将降维流程拆成了两部分：一部分是计算特征空间V，由奇异值分解完成，另一部分是映射数据和求解新特征矩阵，由主成分分析完成，实现了用SVD的性质减少计算量，却让信息量的评估指标是方差，具体流程如下图：



讲到这里，相信大家就能够理解，为什么PCA的类里会包含控制SVD分解器的参数了。通过SVD和PCA的合作，sklearn实现了一种计算更快更简单，但效果却很好的"合作降维"。很多人理解SVD，是把SVD当作PCA的一种求解方法，其实指的就是在矩阵分解时不使用PCA本身的特征值分解，而使用奇异值分解来减少计算量。这种方法确实存在，但在sklearn中，矩阵U和Σ虽然会被计算出来（同样也是一种比起PCA来说简化非常多的数学过程，不产生协方差矩阵），但完全不会被用到，也无法调取查看或者使用，因此我们可以认为，U和Σ在fit过后就被遗弃了。奇异值分解追求的仅仅是V，只要有了V，就可以计算出降维后的特征矩阵。在transform过程之后，fit中奇异值分解的结果除了V(k,n)以外，就会被舍弃，而V(k,n)会被保存在属性components_ 当中，可以调用查看。

```
PCA(2).fit(X).components_

PCA(2).fit(X).components_.shape
```

2.3.2 重要参数svd_solver 与 random_state

参数svd_solver是在降维过程中，用来控制矩阵分解的一些细节的参数。有四种模式可选："auto", "full", "arpack", "randomized"，默认”auto"。

- "**auto**"：基于X.shape和n_components的默认策略来选择分解器：如果输入数据的尺寸大于500x500且要提取的特征数小于数据最小维度min(X.shape)的80%，就启用效率更高的”randomized“方法。否则，精确完整的SVD将被计算，截断将会在矩阵被分解完成后有选择地发生
- "**full**"：从scipy.linalg.svd中调用标准的LAPACK分解器来生成精确完整的SVD，**适合数据量比较适中，计算时间充足的情况**，生成的精确完整的SVD的结构为：

$$U_{(m,m)}, \Sigma_{(m,n)}, V^T_{(n,n)}$$

- "**arpack**"：从scipy.sparse.linalg.svds调用ARPACK分解器来运行截断奇异值分解(SVD truncated)，分解时就将特征数量降到n_components中输入的数值k，**可以加快运算速度，适合特征矩阵很大的时候，但一般用于特征矩阵为稀疏矩阵的情况，**此过程包含一定的随机性。截断后的SVD分解出的结构为：

$$U_{(m,k)}, \Sigma_{(k,k)}, V^T_{(n,n)}$$

- "**randomized**"，通过Halko等人的随机方法进行随机SVD。在"full"方法中，分解器会根据原始数据和输入的n_components值去计算和寻找符合需求的新特征向量，但是在"randomized"方法中，分解器会先生成多个随机向量，然后一一去检测这些随机向量中是否有任何一个符合我们的分解需求，如果符合，就保留这个随机向量，并基于这个随机向量来构建后续的向量空间。这个方法已经被Halko等人证明，比"full"模式下计算快很多，并且还能够保证模型运行效果。**适合特征矩阵巨大，计算量庞大的情况。**

而参数random_state在参数svd_solver的值为"arpack" or "randomized"的时候生效，可以控制这两种SVD模式中的随机模式。通常我们就选用”auto“，不必对这个参数纠结太多。

2.3.3 重要属性components_

现在我们了解了，V(k,n)是新特征空间，是我们要将原始数据进行映射的那些新特征向量组成的矩阵。我们用它来计算新的特征矩阵，但我们希望获取的毕竟是X_dr，为什么我们要把V(k,n)这个矩阵保存在n_components这个属性当中来让大家调取查看呢？

我们之前谈到过PCA与特征选择的区别，即特征选择后的特征矩阵是可解读的，而PCA降维后的特征矩阵式不可解读的：PCA是将已存在的特征进行压缩，降维完毕后的特征不是原本的特征矩阵中的任何一个特征，而是通过某些方式组合起来的新特征。通常来说，**在新的特征矩阵生成之前，我们无法知晓PCA都建立了怎样的新特征向量，新特征矩阵生成之后也不具有可读性**，我们无法判断新特征矩阵的特征是从原数据中的什么特征组合而来，新特征虽然带有原始数据的信息，却已经不是原数据上代表着的含义了。

但是其实，在矩阵分解时，PCA是有目标的：在原有特征的基础上，找出能够让信息尽量聚集的新特征向量。在sklearn使用的PCA和SVD联合的降维方法中，这些新特征向量组成的新特征空间其实就是V(k,n)。当V(k,n)是数字时，我们无法判断V(k,n)和原有的特征究竟有着怎样千丝万缕的数学联系。但是，如果原特征矩阵是图像，V(k,n)这个空间矩阵也可以被可视化的话，我们就可以通过两张图来比较，就可以看出新特征空间究竟从原始数据里提取了什么重要的信息。

让我们来看一个，人脸识别中属性components_的运用。

1. 导入需要的库和模块

```
from sklearn.datasets import fetch_lfw_people
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import numpy as np
```

2. 实例化数据集，探索数据

```
faces = fetch_lfw_people(min_faces_per_person=60)
faces.images.shape
#怎样理解这个数据的维度?
faces.data.shape
#换成特征矩阵之后，这个矩阵是什么样?
X = faces.data
```

3. 看看图像什么样？将原特征矩阵进行可视化

```
#数据本身是图像，和数据本身只是数字，使用的可视化方法不同

#创建画布和子图对象
fig, axes = plt.subplots(4,5
                         ,figsize=(8,4)
                         ,subplot_kw = {"xticks":[],"yticks":[]} #不要显示坐标轴
                         )
fig

axes

#不难发现，axes中的一个对象对应fig中的一个空格
#我们希望，在每一个子图对象中填充图像（共24张图），因此我们需要写一个在子图对象中遍历的循环

axes.shape

#二维结构，可以有两种循环方式，一种是使用索引，循环一次同时生成一列上的三个图
#另一种是把数据拉成一维，循环一次只生成一个图
#在这里，究竟使用哪一种循环方式，是要看我们要画的图的信息，储存在一个怎样的结构里
#我们使用 子图对象.imshow 来将图像填充到空白画布上
#而imshow要求的数据格式必须是一个(m,n)格式的矩阵，即每个数据都是一张单独的图
#因此我们需要遍历的是faces.images，其结构是(1277, 62, 47)
#要从一个数据集中取出24个图，明显是一次性的循环切片[i,:,:]来得便利
#因此我们要把axes的结构拉成一维来循环

axes.flat

enumerate(axes.flat)

#填充图像
for i, ax in enumerate(axes.flat):
    ax.imshow(faces.images[i,:,:]
              ,cmap="gray" #选择色彩的模式
```

```
            )

#https://matplotlib.org/tutorials/colors/colormaps.html
```

4. **建模降维，提取新特征空间矩阵**

```
#原本有2900维，我们现在来降到150维
pca = PCA(150).fit(X)

V = pca.components_
V.shape
```

5. **将新特征空间矩阵可视化**

```
fig, axes = plt.subplots(3,8,figsize=(8,4),subplot_kw = {"xticks":[],"yticks":[]})

for i, ax in enumerate(axes.flat):
    ax.imshow(V[i,:].reshape(62,47),cmap="gray")
```

这张图稍稍有一些恐怖，但可以看出，比起降维前的数据，新特征空间可视化后的人脸非常模糊，这是因为原始数据还没有被映射到特征空间中。但是可以看出，整体比较亮的图片，获取的信息较多，整体比较暗的图片，却只能看见黑漆漆的一块。在比较亮的图片中，眼睛，鼻子，嘴巴，都相对清晰，脸的轮廓，头发之类的比较模糊。

这说明，新特征空间里的特征向量们，大部分是"五官"和"亮度"相关的向量，所以新特征向量上的信息肯定大部分是由原数据中和"五官"和"亮度"相关的特征中提取出来的。到这里，我们通过可视化新特征空间V，解释了一部分降维后的特征：虽然显示出来的数字看着不知所云，但画出来的图表示，这些特征是和”五官“以及”亮度“有关的。这也再次证明了，PCA能够将原始数据集中重要的数据进行聚集。

2.4 重要接口inverse_transform

在上周的特征工程课中，我们学到了神奇的接口inverse_transform，可以将我们归一化，标准化，甚至做过哑变量的特征矩阵还原回原始数据中的特征矩阵，这几乎在向我们暗示，任何有inverse_transform这个接口的过程都是可逆的。PCA应该也是如此。在sklearn中，我们通过让原特征矩阵X右乘新特征空间矩阵V(k,n)来生成新特征矩阵X_dr，那理论上来说，让新特征矩阵X_dr右乘V(k,n)的逆矩阵$V_{(k,n)}^{-1}$，就可以将新特征矩阵X_dr还原为X。那sklearn是否这样做了呢？让我们来看看下面的案例。

2.4.1 迷你案例：用人脸识别看PCA降维后的信息保存量

人脸识别是最容易的，用来探索inverse_transform功能的数据。我们先调用一组人脸数据X(m,n)，对人脸图像进行绘制，然后我们对人脸数据进行降维得到X_dr，之后再使用inverse_transform(X_dr)返回一个X_inverse(m,n)，并对这个新矩阵中的人脸图像也进行绘制。如果PCA的降维过程是可逆的，我们应当期待X(m,n)和X_inverse(m,n)返回一模一样的图像，即携带一模一样的信息。

1. **导入需要的库和模块**(与2.3.3节中步骤一致)

```
from sklearn.datasets import fetch_lfw_people
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import numpy as np
```

2. **导入数据，探索数据**(与2.3.3节中步骤一致)

```
faces = fetch_lfw_people(min_faces_per_person=60)
faces.images.shape
#怎样理解这个数据的维度?
faces.data.shape
#换成特征矩阵之后，这个矩阵是什么样?
X = faces.data
```

3. **建模降维，获取降维后的特征矩阵X_dr**

```
pca = PCA(150)
X_dr = pca.fit_transform(X)
X_dr.shape
```

4. **将降维后矩阵用inverse_transform返回原空间**

```
X_inverse = pca.inverse_transform(X_dr)

X_inverse.shape
```

5. **将特征矩阵X和X_inverse可视化**

```
fig, ax = plt.subplots(2,10,figsize=(10,2.5)
                       ,subplot_kw={"xticks":[],"yticks":[]}
                      )

#和2.3.3节中的案例一样，我们需要对子图对象进行遍历的循环，来将图像填入子图中
#那在这里，我们使用怎样的循环？
#现在我们的ax中是2行10列，第一行是原数据，第二行是inverse_transform后返回的数据
#所以我们需要同时循环两份数据，即一次循环画一列上的两张图，而不是把ax拉平

for i in range(10):
    ax[0,i].imshow(face.image[i,:,:],cmap="binary_r")
    ax[1,i].imshow(X_inverse[i].reshape(62,47),cmap="binary_r")
```

可以明显看出，这两组数据可视化后，由降维后再通过inverse_transform转换回原维度的数据画出的图像和原数据画的图像大致相似，但原数据的图像明显更加清晰。这说明，inverse_transform并没有实现数据的完全逆转。这是因为，在降维的时候，部分信息已经被舍弃了，X_dr中往往不会包含原数据100%的信息，所以在逆转的时候，即便维度升高，原数据中已经被舍弃的信息也不可能再回来了。所以，**降维不是完全可逆的。**

Inverse_transform的功能，是基于X_dr中的数据进行升维，将数据重新映射到原数据所在的特征空间中，而并非恢复所有原有的数据。但同时，我们也可以看出，降维到300以后的数据，的确保留了原数据的大部分信息，所以图像看起来，才会和原数据高度相似，只是稍稍模糊罢了。

2.4.2 迷你案例：用PCA做噪音过滤

降维的目的之一就是希望抛弃掉对模型带来负面影响的特征，而我们相信，带有效信息的特征的方差应该是远大于噪音的，所以相比噪音，有效的特征所带的信息应该不会在PCA过程中被大量抛弃。inverse_transform能够在不恢复原始数据的情况下，将降维后的数据返回到原本的高维空间，即是说能够实现"保证维度，但去掉方差很小特征所带的信息"。利用inverse_transform的这个性质，我们能够实现噪音过滤。

1. **导入所需要的库和模块**

```
from sklearn.datasets import load_digits
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import numpy as np
```

2. **导入数据，探索数据**

```
digits = load_digits()
digits.data.shape
```

3. **定义画图函数**

```
def plot_digits(data):
    fig, axes = plt.subplots(4,10,figsize=(10,4)
                             ,subplot_kw = {"xticks":[],"yticks":[]}
                             )
    for i, ax in enumerate(axes.flat):
        ax.imshow(data[i].reshape(8,8),cmap="binary")

plot_digits(digits.data)
```

4. **为数据加上噪音**

```
np.random.RandomState(42)

#在指定的数据集中，随机抽取服从正态分布的数据
#两个参数，分别是指定的数据集，和抽取出来的正太分布的方差
noisy = np.random.normal(digits.data,2)

plot_digits(noisy)
```

5. **降维**

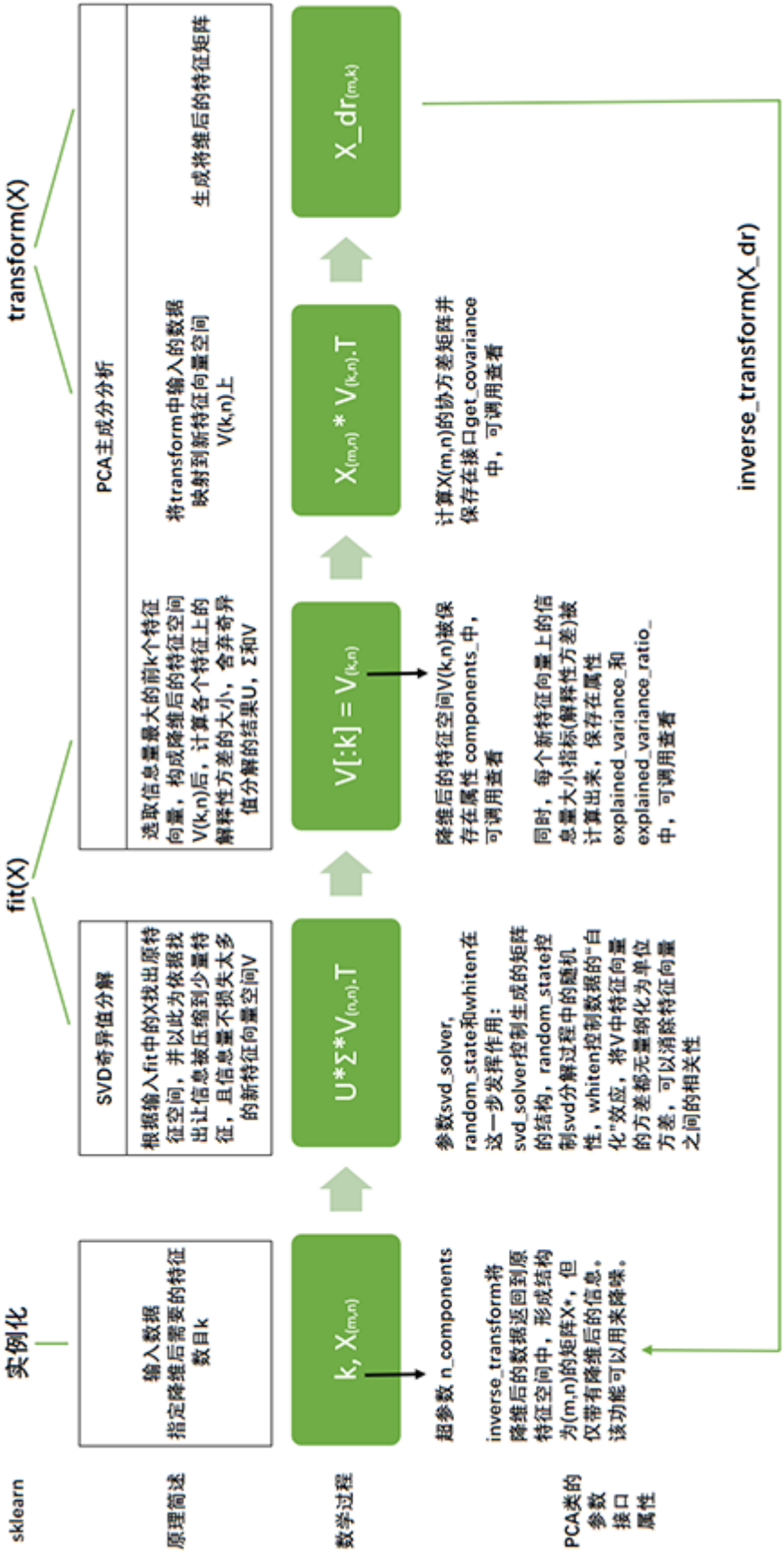
```
pca = PCA(0.5).fit(noisy)
X_dr = pca.transform(noisy)
X_dr.shape
```

6. **逆转降维结果，实现降噪**

```
without_noise = pca.inverse_transform(X_dr)
plot_digits(without_noise)
```

2.5 重要接口，参数和属性总结

到现在，我们已经完成了对PCA的讲解。我们讲解了重要参数参数n_components，svd_solver，random_state，讲解了三个重要属性：components_, explained_variance_以及explained_variance_ratio_，无数次用到了接口fit，transform，fit_transform，还讲解了与众不同的重要接口inverse_transform。所有的这些内容都可以被总结在这张图中：


sklearn
实例化
fit(X)
transform(X)
inverse_transform(X_dr)
原理简述
数学过程
PCA类的参数 接口 属性
输入数据
指定降维后需要的特征数目k
SVD奇异值分解
根据输入fit中的X找出原特征空间，并以此为依据找出让信息被压缩到少量特征，且信息量不损失太多的新特征向量空间V
PCA主成分分析
选取信息量最大的前k个特征向量，构成降维后的特征空间V(k,n)后，计算各个特征上的解释性方差的大小，舍弃奇异值分解的结果U，Σ和V
将transform中输入的数据映射到新特征向量空间V(k,n)上
生成将维后的特征矩阵
k, X(m,n)
U*Σ*V(n,n).T
V[:k] = V(k,n)
X(m,n) * V(k,n).T
X_dr(m,k)
超参数 n_components
参数svd_solver, random_state和whiten在这一步发挥作用：svd_solver控制生成的矩阵的结构，random_state控制svd分解过程中的随机性，whiten控制数据的"白化"效应，将V中特征向量的方差都无量纲化为单位方差，可以消除特征向量之间的相关性
降维后的特征空间V(k,n)被保存在属性 components_中，可调用查看
计算X(m,n)的协方差矩阵并保存在接口get_covariance中，可调用查看
同时，每个新特征向量上的信息量大小指标(解释性方差)被计算出来，保存在属性explained_variance_和explained_variance_ratio_中，可调用查看
inverse_transform将降维后的数据返回到原特征空间中，形成结构为(m,n)的矩阵X*，但仅带有降维后的信息。该功能可以用来降噪。

3 案例：PCA对手写数字数据集的降维

还记得我们上一周在讲特征工程时，使用的手写数字的数据集吗？数据集结构为(42000, 784)，用KNN跑一次半小时，得到准确率在96.6%上下，用随机森林跑一次12秒，准确率在93.8%，虽然KNN效果好，但由于数据量太大，KNN计算太缓慢，所以我们不得不选用随机森林。我们使用了各种技术对手写数据集进行特征选择，最后使用嵌入法SelectFromModel选出了324个特征，将随机森林的效果也调到了96%以上。但是，因为数据量依然巨大，还是有300多个特征。今天，我们就来试着用PCA处理一下这个数据，看看效果如何。

1. **导入需要的模块和库**

```
from sklearn.decomposition import PCA
from sklearn.ensemble import RandomForestClassifier as RFC
from sklearn.model_selection import cross_val_score
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
```

2. **导入数据，探索数据**

```
data = pd.read_csv(r"C:\work\learnbetter\micro-class\week 3 Preprocessing\digit recognizor.csv")

X = data.iloc[:,1:]
y = data.iloc[:,0]

X.shape
```

3. **画累计方差贡献率曲线，找最佳降维后维度的范围**

```
pca_line = PCA().fit(X)
plt.figure(figsize=[20,5])
plt.plot(np.cumsum(pca_line.explained_variance_ratio_))
plt.xlabel("number of components after dimension reduction")
plt.ylabel("cumulative explained variance ratio")
plt.show()
```

4. **降维后维度的学习曲线，继续缩小最佳维度的范围**

```
#======【TIME WARNING: 2mins 30s】======#

score = []
for i in range(1,101,10):
    X_dr = PCA(i).fit_transform(X)
    once = cross_val_score(RFC(n_estimators=10,random_state=0)
                           ,X_dr,y,cv=5).mean()
    score.append(once)
plt.figure(figsize=[20,5])
plt.plot(range(1,101,10),score)
plt.show()
```

5. **细化学习曲线，找出降维后的最佳维度**

```
#======【TIME WARNING: 2mins 30s】======#

score = []
for i in range(10,25):
    X_dr = PCA(i).fit_transform(X)
    once = cross_val_score(RFC(n_estimators=10,random_state=0),X_dr,y,cv=5).mean()
    score.append(once)
plt.figure(figsize=[20,5])
plt.plot(range(10,25),score)
plt.show()
```

6. **导入找出的最佳维度进行降维，查看模型效果**

```
X_dr = PCA(23).fit_transform(X)

#======【TIME WARNING:1mins 30s】======#
cross_val_score(RFC(n_estimators=100,random_state=0),X_dr,y,cv=5).mean()
```

模型效果还好，跑出了94.49%的水平，但还是没有我们使用嵌入法特征选择过后的96%高，有没有什么办法能够提高模型的表现呢？

7. **突发奇想，特征数量已经不足原来的3%，换模型怎么样？**

在之前的建模过程中，因为计算量太大，所以我们一直使用随机森林，但事实上，我们知道KNN的效果比随机森林更好，KNN在未调参的状况下已经达到96%的准确率，而随机森林在未调参前只能达到93%，这是模型本身的限制带来的，这个数据使用KNN效果就是会更好。现在我们的特征数量已经降到不足原来的3%，可以使用KNN了吗？

```
from sklearn.neighbors import KNeighborsClassifier as KNN
cross_val_score(KNN(),X_dr,y,cv=5).mean()
```

8. **KNN的k值学习曲线**

```
#======【TIME WARNING: 】======#

score = []
for i in range(10):
    X_dr = PCA(23).fit_transform(X)
    once = cross_val_score(KNN(i+1),X_dr,y,cv=5).mean()
    score.append(once)
plt.figure(figsize=[20,5])
plt.plot(range(10),score)
plt.show()
```

9. **定下超参数后，模型效果如何，模型运行时间如何？**

```
cross_val_score(KNN(4),X_dr,y,cv=5).mean()


#=======【TIME WARNING: 3mins】======#
%%timeit
cross_val_score(KNN(4),X_dr,y,cv=5).mean()
```

可以发现，原本785列的特征被我们缩减到23列之后，用KNN跑出了目前位置这个数据集上最好的结果。再进行更细致的调整，我们也许可以将KNN的效果调整到98%以上。PCA为我们提供了无限的可能，终于不用再因为数据量太庞大而被迫选择更加复杂的模型了！

4 附录

4.1 PCA参数列表

n_components	整数，浮点数，None或输入字符串 要保留的特征数量。若不填写，则保留的特征数量为数据最小维度min(X.shape)。 - 输入"mle"并且参数svd_solver =="full"，表示使用Minka, T.P. 的自动选择PCA维度法来猜测维度，此论文在NIPS上，第598-604页。注意，输入"mle"参数后svd_solver =='auto'也会被理解为svd_solver =='full'。 - 输入[0,1]之间的浮点数且svd_solver =='full'，表示选择让"需要解释的总方差量"大于n_components指定的数目的那些特征 - 输入"None"，则保留的特征数量为数据最小维度-1，即min(X.shape) - 1 如果svd_solver =='arpack'，则特征数必须严格小于数据最小维度min(X.shape)
copy	布尔值，可不填，默认True 如果为False，则传递给fit的数据将被覆盖并且运行fit(X).transform(X)将不会产生预期结果，请改用fit_transform(X)
whiten	布尔值，可不填，默认False 控制输入PCA的特征矩阵的白化。白化是数据预处理的一种，其目的是去掉特征与特征之间的相关性，并将所有特征的方差都归一化。当为True时，components_(即奇异值分解中分解出来的矩阵v)中的向量会被乘以样本数量的平方根sqrt(n_samples)，然后除以奇异值(奇异矩阵Σ的对角线上的元素Σi)，以确保特征向量之间不相关并且每个特征都具有单位方差。 白化将从变换后的信号中去除一些信息（比如特征之间的相对方差量纲），但有时可以通过使数据遵循一些硬连线假设来提高下游估计器的预测精度。

svd_solver	输入字符串，可选"auto", "full", "arpack", "randomized" 控制SVD奇异值分解的分解模式，默认 - 输入"auto"：基于X.shape和n_components的默认策略来选择分解器：如果输入数据的尺寸大于500x500且要提取的特征数小于数据最小维度min(X.shape)的80%，就启用效率更高的'随机化'方法。否则，精确完整的SVD将被计算，截断将会在矩阵被分解完成后有选择地发生。 - 输入"full"：运行精确完整的SVD，从scipy.linalg.svd中调用标准的LAPACK分解器，并通过后处理来选择特征。完整的SVD的结构为：U(m,m), S(m,n), V(n,n)。 - 输入"arpack"：从scipy.sparse.linalg.svds调用ARPACK分解器来运行截断奇异值分解(SVD truncated)，以在分解时就将特征数量降到n_components中输入的数值，严格要求n_components的取值在区间[0,数据最小维度min(X.shape)]之间。截断后的SVD分解出的结构为：U(m,n_components), S(n_components, n_componenst), V(n,n)。 - 输入"randomized"，通过Halko等人的随机方法进行随机SVD。在"full"方法中，分解器会根据原始数据和输入的n_components值去计算和寻找符合需求的新特征向量，但是在"randomized"方法中，分解器会先生成多个随机向量，然后一一去检测这些随机向量中是否有任何一个符合我们的分解需求，如果符合，就保留这个随机向量，并基于这个随机向量来构建后续的向量空间。这个方法已经被Halko等人证明，比"full"模式下计算快很多，并且还能够保证模型运行效果。 具体请参考Halko, N.，Martinsson, P.G.以及Tropp, J.A.于2011年发表的论文《寻找具有随机性的结构：用于构造近似矩阵分解的概率算法》（*Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions*），来自学术期刊SIAM review, 53期第2册, 217-288页，请同时参考Martinsson, P. G., Rokhlin, V., and Tygert, M. 于2011年发表的《用于矩阵分解的随机算法》（*A randomized algorithm for the decomposition of matrices*），来自学术刊物Applied and Computational Harmonic Analysis, 30期第1册, 47-68页。

tol	大于等于0的浮点数，默认为.0 当svd_solver为"arpack"的时候，计算奇异值所需要的容差。 *0.18.0版本中新增的功能。*
iterated_power	输入大于0的整数，或者"auto"，默认"auto" 当svd_solver的时候'randomized'，计算的幂方法的迭代次数。 *0.18.0版本中新增的功能。*
random_state	整数，sklearn中设定好的RandomState实例，或None，可不填，默认None 仅当参数svd_solver的值为"arpack" or "randomized"的时候才有效。 1）输入整数，random_state是由随机数生成器生成的随机数种子 2）输入RandomState实例，则random_state是一个随机数生成器 3）输入None，随机数生成器会是np.random模块中的一个RandomState实例 0.18.0版本中新增的功能。

4.2 PCA属性列表

components_	数组，结构为(n_components, n_features) 新特征空间V中的特征向量的方向，按explained_variance_的大小排序
explained_variance_	数组，结构为(n_components,) 每个所选特征的解释性方差 等于X的协方差矩阵的前n_components个最大特征值 *0.18.0版本中新增的功能*
explained_variance_ratio_	数组，结构为(n_components,) 每个所选特征的解释性方差占原数据方差综合的百分比 如果没有输入任何n_components，则保留所有的特征，并且比率综合为1
singular_values_	数组，结构为(n_components,) 返回每个所选特征的奇异值。奇异值等于低维空间中n_components变量的2范式
mean_	数组，结构为(n_features,) 从训练集估计出的每个特征的均值，数值上等于 X.mean(axis=0)
n_components_	整数 估计的特征数量。当参数n_components被设置为mle或0~1之间的数字时(并且svd_solver被设置为"full")，将根据输入的特征矩阵估算降维后的特征个数。否则，这个属性会等于输入参数n_components中的值。如果参数n_components被设置为None，这个属性会返回min(X.shape)，即数据的最小维度。
noise_variance_	浮点数 根据1999年Tipping和Bishop的Probabilistic PCA模型估计的噪声协方差。参见C.Bishop论文*《模式识别和机器学习》*，章节12.2.1 p.574或http://www.miketipping.com/papers/met-mppca.pdf。需要计算估计的数据协方差和分数样本。 等于特征矩阵的协方差矩阵的（min(X.shape) - n_components）的最小特征值的平均值。

4.3 PCA接口列表

接口	输入	含义	返回
fit	特征矩阵X	使用特征矩阵拟合模型	拟合好的模型本身
transform	特征矩阵X	将降维应用到特征矩阵	降维后的特征矩阵
fit_transform	特征矩阵X	使用特征矩阵拟合模型并且在特征矩阵上应用降维	降维后的特征矩阵
inverse_transform	特征矩阵X	将数据转换回原始空间	返回到原始空间的矩阵，该矩阵与原始特征矩阵结构相同，但数据不同
score	特征矩阵X	返回所有样本的平均对数似然	参见C.Bishop论文《模式识别和机器学习》，章节12.2.1 p.574或http://www.miketipping.com/papers/met-mppca.pdf。
score_samples	不需要输入任何对象	返回每个样本的对数似然	
get_covariance	不需要输入任何对象	在生成的模型上计算数据的协方差矩阵	数据的协方差 协方差的公式为： cov = components_.T * S^2 * components_ + sigma2 * eye(n_features) 其中S^2包含解释的方差，sigma2包含噪声方差。
get_precision	不需要输入任何对象	在生成的模型上计算数据的精度矩阵	返回估计数据的精确度。 等于协方差的倒数，但用矩阵求逆引理来提升计算效率。
get_params	不需要输入任何对象	获取此评估器的参数	模型的参数
set_params	新参数组合	在建立好的模型上，重新设置此评估器的参数	用新参数组合重新实例化和训练的模型