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。

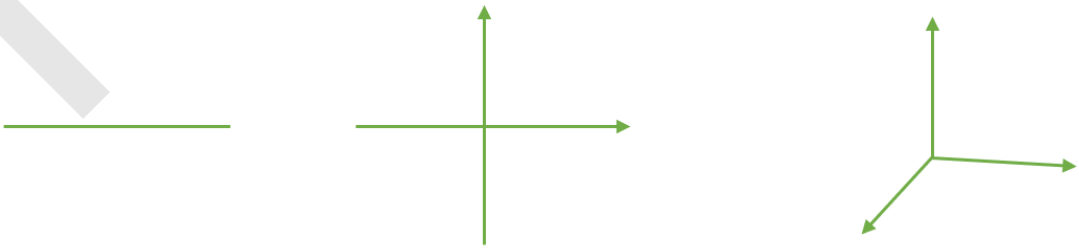
		特征1
DataFrame 特征矩阵	0	0.0
	1	0.0

	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。

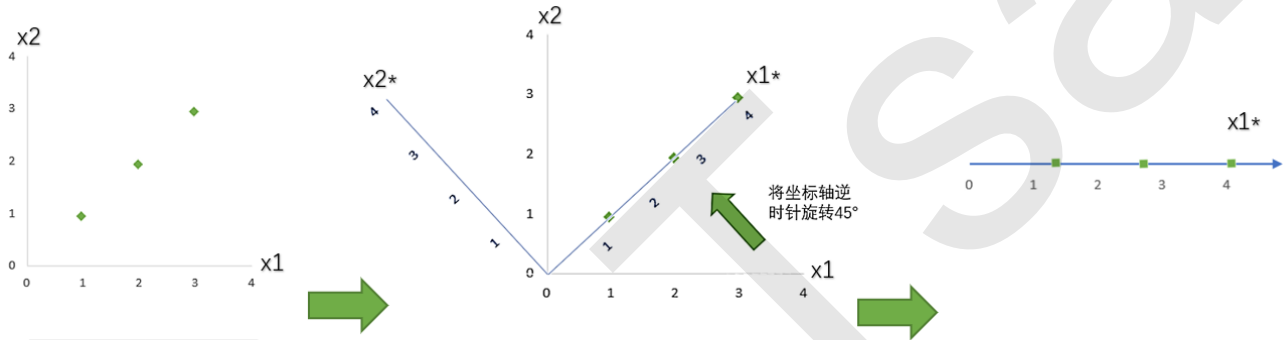
面试高危问题

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：

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

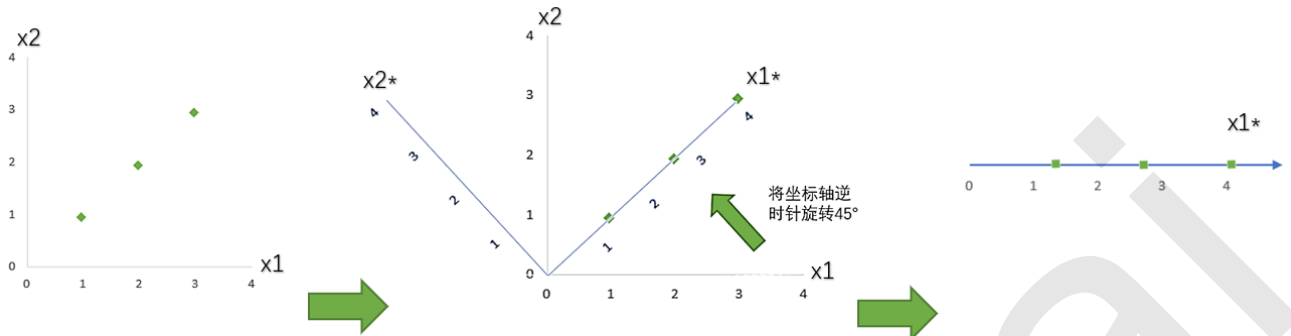
每个特征的数据一模一样，因此方差也都为1，数据的方差总和是2。

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，x2*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
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
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：降维和特征选择都是特征工程技术，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。

而降维算法，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，我们无法知晓降维算法们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。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")
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 重要参数svd_solver

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}）。

$$传统SVD：\ X \rightarrow 数学神秘的宇宙 \rightarrow U\Sigma V^T$$

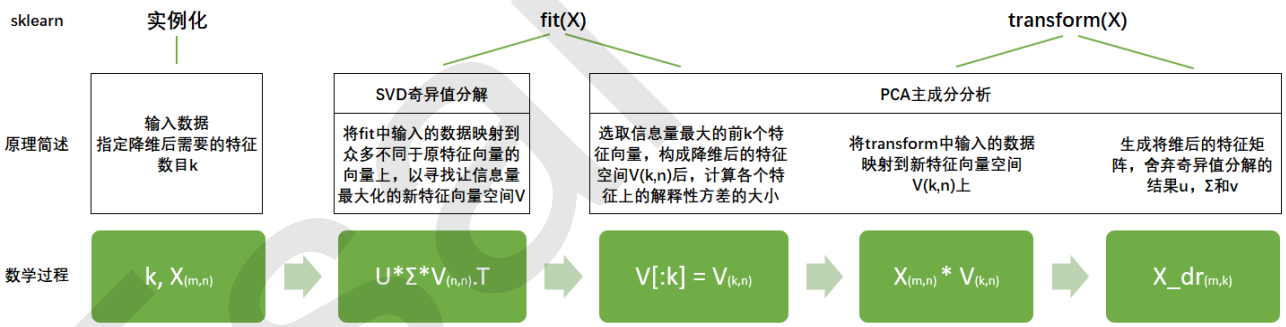
$$开挂的SVD：\ X \rightarrow 一个简化非常多的数学过程 \rightarrow V^T$$

右奇异矩阵V^{T}有着如下性质：

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

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)}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。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。

【完整版】2.3.2 svd_solver 与 random_state

【完整版】2.4 重要接口inverse_transform

【完整版】2.4.1 神奇的inverse_transform

【完整版】2.4.2 迷你案例：用人脸识别看PCA降维后的信息保存量

【完整版】2.4.3 迷你案例：用PCA做噪音过滤

3 直播小专题：究竟从哪里开始机器学习?

- **学习机器学习需要什么基础?**

  Python或R，至少要熟练一个

- **数学不好行不行?**

  现在不好可以，但数学永远是过不去的坎：统计学，线性代数，概率论，不能永远不好

  对自己的数学没信心？我也数学苦手，但是慢慢琢磨，总有会的一天

- **英语不好行不行?**

  行，这个真行。

- **实践重要还是理论更重要?**

  工作中实践重要，面试的时候两个都重要。去面试时，做业务的可能会死命问你原理，做技术的可能会死命问代码，但是无论什么岗位，熟练掌握总是没错。

- **转行行不行?**

  行，我自己就是转行，以前是做金融的。但是这是个长期的过程，数学和算法都不可能抱佛脚抱出来，要好好累积好好打磨自己，短时间之内成为算法大神是不可能的，除非你是以前未曾被挖掘出来的算法天才。在全职工作并且工作本身与数据无关的前提下，我建议你至少刻苦学习半年以上再转。

- **该怎么学习?**

  小白入门，你需要至少一本体系完整的书或一门体系完整的课，来打基础。

  可以是吴恩达大神的课，他的课免费，理论清楚，高屋建瓴，不过是全英文的而且中文翻译比较坑，还有就是年带比较早了，大概是09年左右录的。

  理论：《数据挖掘导论》，没有更好的了，能读英文的最好，不能读英文的，建议中英双语一起实用

  数学：考研三部曲，据说国内上过大学的应该都学过，我自己没学过，回头给大家找图补上

  Python：《Python数据科学手册》，封面上一只蜥蜴

  打完基础之后，你就可以自学了，网上有众多大神写的博客，有我们这样的公开课，还有进阶书单：

★ 数据科学

Facebook 数据科学家作品

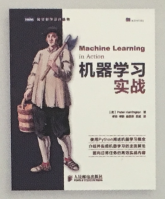
零基础上手

Amazon 畅销书

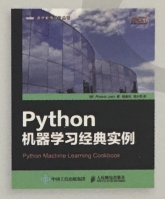
详解第三方库与工具

★ 机器学习与深度学习

机器学习入门首选



机器学习畅销经典

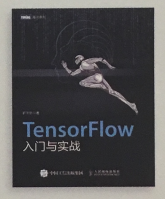


Kaggle 经典数据集和案例

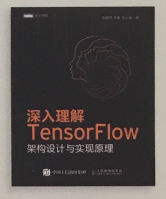
Bing 核心团队成员作品

你的第一本深度学习书

高清彩图辅助理解

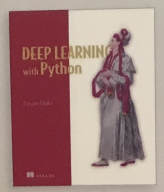


一线人员力作

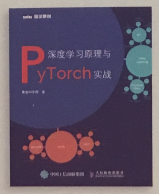


深度学习进阶必备

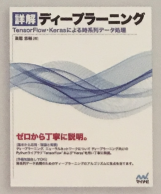
★ 重点新书预告



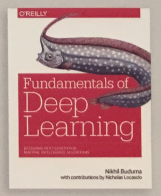
Keras 之父作品



通过 PyTorch 入门深度学习



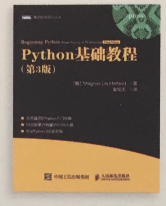
日亚深度学习畅销书



TensorFlow 入门

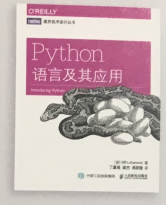
★ 编程入门

零基础入门



有其他编程语言基础入门

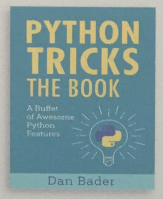
注重计算思维



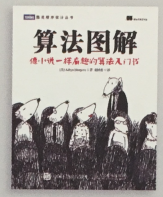
有其他编程语言基础上手

★ 编程进阶

进阶必备经典



进阶必备，正在翻译



超轻松学习实用算法

豆瓣评分 9.2

Flask 第一书

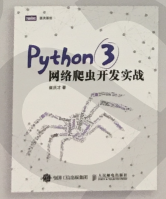
Amazon 全五星好评

深入浅出，易学易用

韩国 Python 安全经典

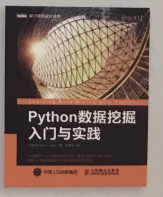
★ 网络爬虫与数据挖掘

超受欢迎爬虫入门



百万访问量
博客作者静觅作品

详细剖析数据挖掘算法



掌握数据挖掘最佳实践