## [强大的矩阵奇异值分解(SVD)及其应用](http://www.cnblogs.com/liangflying/archive/2012/09/25/2701148.html)

**版权声明：**

    本文由LeftNotEasy发布于[http://leftnoteasy.cnblogs.com](http://leftnoteasy.cnblogs.com/), 本文可以被全部的转载或者部分使用，但请注明出处，如果有问题，请联系[wheeleast@gmail.com](mailto:wheeleast@gmail.com)

**前言：**

    上一次写了关于[PCA与LDA](http://www.cnblogs.com/LeftNotEasy/archive/2011/01/08/lda-and-pca-machine-learning.html)的文章，PCA的实现一般有两种，一种是用特征值分解去实现的，一种是用奇异值分解去实现的。在上篇文章中便是基于特征值分解的一种解释。特征值和奇异值在大部分人的印象中，往往是停留在纯粹的数学计算中。而且线性代数或者矩阵论里面，也很少讲任何跟特征值与奇异值有关的应用背景。奇异值分解是一个有着很明显的物理意义的一种方法，它可以将一个比较复杂的矩阵用更小更简单的几个子矩阵的相乘来表示，这些小矩阵描述的是矩阵的重要的特性。就像是描述一个人一样，给别人描述说这个人长得浓眉大眼，方脸，络腮胡，而且带个黑框的眼镜，这样寥寥的几个特征，就让别人脑海里面就有一个较为清楚的认识，实际上，人脸上的特征是有着无数种的，之所以能这么描述，是因为人天生就有着非常好的抽取重要特征的能力，让机器学会抽取重要的特征，SVD是一个重要的方法。

    在机器学习领域，有相当多的应用与奇异值都可以扯上关系，比如做feature reduction的PCA，做数据压缩（以图像压缩为代表）的算法，还有做搜索引擎语义层次检索的LSI（Latent Semantic Indexing）

    另外在这里抱怨一下，之前在百度里面搜索过SVD，出来的结果都是俄罗斯的一种狙击枪（AK47同时代的），是因为穿越火线这个游戏里面有一把狙击枪叫做SVD，而在Google上面搜索的时候，出来的都是奇异值分解（英文资料为主）。想玩玩战争游戏，玩玩COD不是非常好吗，玩山寨的CS有神马意思啊。国内的网页中的话语权也被这些没有太多营养的帖子所占据。真心希望国内的气氛能够更浓一点，搞游戏的人真正是喜欢制作游戏，搞Data Mining的人是真正喜欢挖数据的，都不是仅仅为了混口饭吃，这样谈超越别人才有意义，中文文章中，能踏踏实实谈谈技术的太少了，改变这个状况，从我自己做起吧。

    前面说了这么多，本文主要关注奇异值的一些特性，另外还会稍稍提及奇异值的计算，不过本文不准备在如何计算奇异值上展开太多。另外，本文里面有部分不算太深的线性代数的知识，如果完全忘记了线性代数，看本文可能会有些困难。

**一、奇异值与特征值基础知识：**

    特征值分解和奇异值分解在机器学习领域都是属于满地可见的方法。两者有着很紧密的关系，我在接下来会谈到，特征值分解和奇异值分解的目的都是一样，就是提取出一个矩阵最重要的特征。先谈谈特征值分解吧：

   1）**特征值：**

    如果说一个向量v是方阵A的特征向量，将一定可以表示成下面的形式：

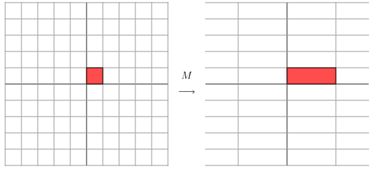
[image](http://images.cnblogs.com/cnblogs_com/LeftNotEasy/201101/201101192226321862.png)

    这时候λ就被称为特征向量v对应的特征值，一个矩阵的一组特征向量是一组正交向量。特征值分解是将一个矩阵分解成下面的形式：

[image](http://images.cnblogs.com/cnblogs_com/LeftNotEasy/201101/201101192226321023.png)

    其中Q是这个矩阵A的特征向量组成的矩阵，Σ是一个对角阵，每一个对角线上的元素就是一个特征值。我这里引用了一些参考文献中的内容来说明一下。首先，要明确的是，一个矩阵其实就是一个线性变换，因为一个矩阵乘以一个向量后得到的向量，其实就相当于将这个向量进行了线性变换。比如说下面的一个矩阵：

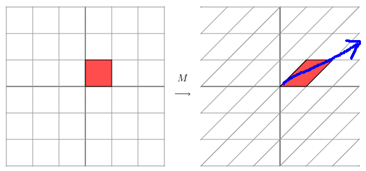
[image](http://images.cnblogs.com/cnblogs_com/LeftNotEasy/201101/20110119222632500.png)    它其实对应的线性变换是下面的形式：

[](http://images.cnblogs.com/cnblogs_com/LeftNotEasy/201101/201101192226326073.png)    因为这个矩阵M乘以一个向量(x,y)的结果是：

[image](http://images.cnblogs.com/cnblogs_com/LeftNotEasy/201101/201101192226334470.png)    上面的矩阵是对称的，所以这个变换是一个对x，y轴的方向一个拉伸变换（每一个对角线上的元素将会对一个维度进行拉伸变换，当值>1时，是拉长，当值<1时时缩短），当矩阵不是对称的时候，假如说矩阵是下面的样子：

[image](http://images.cnblogs.com/cnblogs_com/LeftNotEasy/201101/201101192226337534.png)

    它所描述的变换是下面的样子：

[](http://images.cnblogs.com/cnblogs_com/LeftNotEasy/201101/201101192226333107.png)

    这其实是在平面上对一个轴进行的拉伸变换（如蓝色的箭头所示），在图中，蓝色的箭头是一个最**主要的**变化方向（变化方向可能有不止一个），**如果我们想要描述好一个变换，那我们就描述好这个变换主要的变化方向就好了**。反过头来看看之前特征值分解的式子，分解得到的Σ矩阵是一个对角阵，里面的特征值是由大到小排列的，这些特征值所对应的特征向量就是描述这个矩阵变化方向（从主要的变化到次要的变化排列）

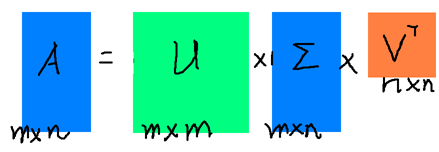
    当矩阵是高维的情况下，那么这个矩阵就是高维空间下的一个线性变换，这个线性变化可能没法通过图片来表示，但是可以想象，这个变换也同样有很多的变换方向，我们通过特征值分解得到的前N个特征向量，那么就对应了这个矩阵最主要的N个变化方向。我们利用这前N个变化方向，就可以近似这个矩阵（变换）。也就是之前说的：**提取这个矩阵最重要的特征。**总结一下，特征值分解可以得到特征值与特征向量，特征值表示的是这个特征到底有多重要，而特征向量表示这个特征是什么，可以将每一个特征向量理解为一个线性的子空间，我们可以利用这些线性的子空间干很多的事情。不过，**特征值分解也有很多的局限，比如说变换的矩阵必须是方阵。**

   （说了这么多特征值变换，不知道有没有说清楚，请各位多提提意见。）

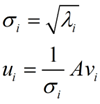
**2）奇异值：**

    下面谈谈奇异值分解。特征值分解是一个提取矩阵特征很不错的方法，但是它只是对方阵而言的，在现实的世界中，我们看到的大部分矩阵都不是方阵，比如说有N个学生，每个学生有M科成绩，这样形成的一个N \* M的矩阵就不可能是方阵，**我们怎样才能描述这样普通的矩阵呢的重要特征呢？**奇异值分解可以用来干这个事情，**奇异值分解是一个能适用于任意的矩阵的一种分解的方法**：

[image](http://images.cnblogs.com/cnblogs_com/LeftNotEasy/201101/201101192226335092.png)    假设A是一个N \* M的矩阵，那么得到的U是一个N \* N的方阵（里面的向量是正交的，U里面的向量称为左奇异向量），Σ是一个N \* M的矩阵（除了对角线的元素都是0，对角线上的元素称为奇异值），V’(V的转置)是一个N \* N的矩阵，里面的向量也是正交的，V里面的向量称为右奇异向量），从图片来反映几个相乘的矩阵的大小可得下面的图片

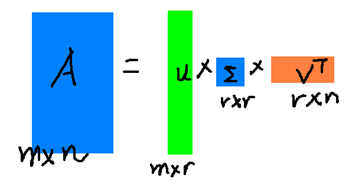
[](http://images.cnblogs.com/cnblogs_com/LeftNotEasy/201101/201101192226341537.png)

    那么奇异值和特征值是怎么对应起来的呢？首先，我们将一个矩阵A的转置 \* A，将会得到一个方阵，我们用这个方阵求特征值可以得到：[image](http://images.cnblogs.com/cnblogs_com/LeftNotEasy/201101/201101192226349618.png)    这里得到的v，就是我们上面的右奇异向量。此外我们还可以得到：

[](http://images.cnblogs.com/cnblogs_com/LeftNotEasy/201101/201101192226344111.png)    这里的σ就是上面说的奇异值，u就是上面说的左奇异向量。奇异值σ跟特征值类似，在矩阵Σ中也是从大到小排列，而且σ的减少特别的快，**在很多情况下，前10%甚至1%的奇异值的和就占了全部的奇异值之和的99%以上了**。也就是说，我们也可以用前r大的奇异值来近似描述矩阵，这里定义一下**部分奇异值分解**：

[image](http://images.cnblogs.com/cnblogs_com/LeftNotEasy/201101/201101192226359684.png)

    r是一个远小于m、n的数，这样矩阵的乘法看起来像是下面的样子：

[](http://images.cnblogs.com/cnblogs_com/LeftNotEasy/201101/201101192226359717.png)

    右边的三个矩阵相乘的结果将会是一个接近于A的矩阵，在这儿，r越接近于n，则相乘的结果越接近于A。而这三个矩阵的面积之和（在存储观点来说，矩阵面积越小，存储量就越小）要远远小于原始的矩阵A，我们如果想要压缩空间来表示原矩阵A，我们存下这里的三个矩阵：U、Σ、V就好了。

**二、奇异值的计算：**

    奇异值的计算是一个难题，是一个O(N^3)的算法。在单机的情况下当然是没问题的，matlab在一秒钟内就可以算出1000 \* 1000的矩阵的所有奇异值，但是当矩阵的规模增长的时候，计算的复杂度呈3次方增长，就需要并行计算参与了。Google的**吴军**老师在**数学之美**系列谈到SVD的时候，说起Google实现了SVD的并行化算法，说这是对人类的一个贡献，但是也没有给出具体的计算规模，也没有给出太多有价值的信息。

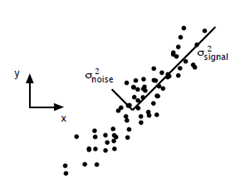
    其实SVD还是可以用并行的方式去实现的，在解大规模的矩阵的时候，一般使用迭代的方法，当矩阵的规模很大（比如说上亿）的时候，迭代的次数也可能会上亿次，如果使用Map-Reduce框架去解，则每次Map-Reduce完成的时候，都会涉及到写文件、读文件的操作。个人猜测Google云计算体系中除了Map-Reduce以外应该还有类似于MPI的计算模型，也就是节点之间是保持通信，数据是常驻在内存中的，这种计算模型比Map-Reduce在解决迭代次数非常多的时候，要快了很多倍。

[Lanczos迭代](http://en.wikipedia.org/wiki/Lanczos_algorithm)就是一种解**对称方阵部分特征值**的方法（之前谈到了，解A’\* A得到的对称方阵的特征值就是解A的右奇异向量），是将一个对称的方程化为一个三对角矩阵再进行求解。按网上的一些文献来看，Google应该是用这种方法去做的奇异值分解的。请见Wikipedia上面的一些引用的论文，如果理解了那些论文，也“几乎”可以做出一个SVD了。

    由于奇异值的计算是一个很枯燥，纯数学的过程，而且前人的研究成果（论文中）几乎已经把整个程序的流程图给出来了。更多的关于奇异值计算的部分，将在后面的参考文献中给出，这里不再深入，我还是focus在奇异值的应用中去。

**三、奇异值与主成分分析（PCA）：**

     主成分分析在上一节里面也讲了一些，这里主要谈谈如何用SVD去解PCA的问题。PCA的问题其实是一个基的变换，使得变换后的数据有着最大的方差。方差的大小描述的是一个变量的信息量，我们在讲一个东西的稳定性的时候，往往说要减小方差，如果一个模型的方差很大，那就说明模型不稳定了。但是对于我们用于机器学习的数据（主要是训练数据），方差大才有意义，不然输入的数据都是同一个点，那方差就为0了，这样输入的多个数据就等同于一个数据了。以下面这张图为例子：

[](http://images.cnblogs.com/cnblogs_com/LeftNotEasy/201101/201101192226359750.png)     这个假设是一个摄像机采集一个物体运动得到的图片，上面的点表示物体运动的位置，假如我们想要用一条直线去拟合这些点，那我们会选择什么方向的线呢？当然是图上标有signal的那条线。如果我们把这些点单纯的投影到x轴或者y轴上，最后在x轴与y轴上得到的方差是相似的（因为这些点的趋势是在45度左右的方向，所以投影到x轴或者y轴上都是类似的），如果我们使用原来的xy坐标系去看这些点，容易看不出来这些点真正的方向是什么。但是如果我们进行坐标系的变化，横轴变成了signal的方向，纵轴变成了noise的方向，则就很容易发现什么方向的方差大，什么方向的方差小了。

    一般来说，方差大的方向是信号的方向，方差小的方向是噪声的方向，我们在数据挖掘中或者数字信号处理中，往往要提高信号与噪声的比例，也就是信噪比。对上图来说，如果我们只保留signal方向的数据，也可以对原数据进行不错的近似了。

    PCA的全部工作简单点说，就是对原始的空间中顺序地找一组相互正交的坐标轴，第一个轴是使得方差最大的，第二个轴是在与第一个轴正交的平面中使得方差最大的，第三个轴是在与第1、2个轴正交的平面中方差最大的，这样假设在N维空间中，我们可以找到N个这样的坐标轴，我们取前r个去近似这个空间，这样就从一个N维的空间压缩到r维的空间了，但是我们选择的r个坐标轴能够使得空间的压缩使得数据的损失最小。

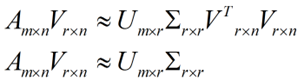
    还是假设我们矩阵每一行表示一个样本，每一列表示一个feature，用矩阵的语言来表示，将一个m \* n的矩阵A的进行坐标轴的变化，P就是一个变换的矩阵从一个N维的空间变换到另一个N维的空间，在空间中就会进行一些类似于旋转、拉伸的变化。

[image](http://images.cnblogs.com/cnblogs_com/LeftNotEasy/201101/201101192226367831.png)

    而将一个m \* n的矩阵A变换成一个m \* r的矩阵，这样就会使得本来有n个feature的，变成了有r个feature了（r < n)，这r个其实就是对n个feature的一种提炼，我们就把这个称为feature的压缩。用数学语言表示就是：

[image](http://images.cnblogs.com/cnblogs_com/LeftNotEasy/201101/201101192226361452.png)    但是这个怎么和SVD扯上关系呢？之前谈到，SVD得出的奇异向量也是从奇异值由大到小排列的，按PCA的观点来看，就是方差最大的坐标轴就是第一个奇异向量，方差次大的坐标轴就是第二个奇异向量…我们回忆一下之前得到的SVD式子：

[image](http://images.cnblogs.com/cnblogs_com/LeftNotEasy/201101/201101192226363437.png)     在矩阵的两边同时乘上一个矩阵V，由于V是一个正交的矩阵，所以V转置乘以V得到单位阵I，所以可以化成后面的式子

[](http://images.cnblogs.com/cnblogs_com/LeftNotEasy/201101/201101192226365422.png)     将后面的式子与A \* P那个m \* n的矩阵变换为m \* r的矩阵的式子对照看看，在这里，其实V就是P，也就是一个变化的向量。这里是将一个m \* n 的矩阵压缩到一个m \* r的矩阵，也就是对列进行压缩，如果我们想对行进行压缩（在PCA的观点下，对行进行压缩可以理解为，将一些相似的sample合并在一起，或者将一些没有太大价值的sample去掉）怎么办呢？同样我们写出一个通用的行压缩例子：

[image](http://images.cnblogs.com/cnblogs_com/LeftNotEasy/201101/201101192226375455.png)    这样就从一个m行的矩阵压缩到一个r行的矩阵了，对SVD来说也是一样的，我们对SVD分解的式子两边乘以U的转置U'

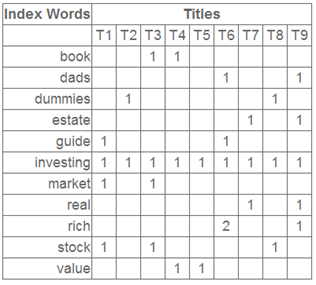
[image](http://images.cnblogs.com/cnblogs_com/LeftNotEasy/201101/201101192226377440.png)    这样我们就得到了对行进行压缩的式子。可以看出，其实PCA几乎可以说是对SVD的一个包装，如果我们实现了SVD，那也就实现了PCA了，而且更好的地方是，有了SVD，我们就可以得到两个方向的PCA，如果我们对A’A进行特征值的分解，只能得到一个方向的PCA。

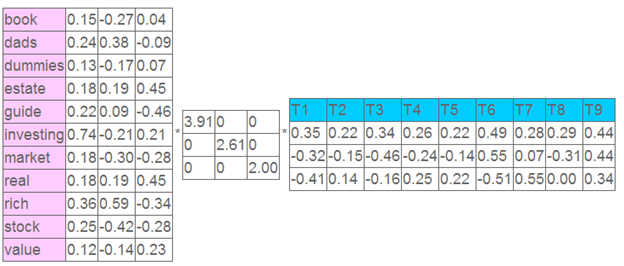
**四、奇异值与潜在语义索引LSI：**

     潜在语义索引（Latent Semantic Indexing）与PCA不太一样，至少不是实现了SVD就可以直接用的，不过LSI也是一个严重依赖于SVD的算法，之前吴军老师在[矩阵计算与文本处理中的分类问题](http://www.google.com.hk/ggblog/googlechinablog/2006/12/blog-post_8935.html)中谈到：

*“三个矩阵有非常清楚的物理含义。第一个矩阵X中的每一行表示意思相关的一类词，其中的每个非零元素表示这类词中每个词的重要性（或者说相关性），数值越大越相关。最后一个矩阵Y中的每一列表示同一主题一类文章，其中每个元素表示这类文章中每篇文章的相关性。中间的矩阵则表示类词和文章雷之间的相关性。因此，我们只要对关联矩阵A进行一次奇异值分解，w 我们就可以同时完成了近义词分类和文章的分类。（同时得到每类文章和每类词的相关性）。”*

     上面这段话可能不太容易理解，不过这就是LSI的精髓内容，我下面举一个例子来说明一下，下面的例子来自LSA tutorial，具体的网址我将在最后的引用中给出：

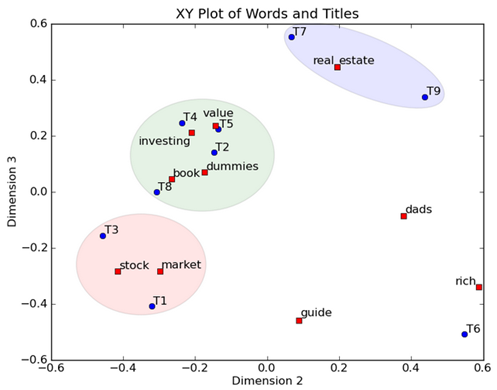
[](http://images.cnblogs.com/cnblogs_com/LeftNotEasy/201101/201101192226379109.png)      这就是一个矩阵，不过不太一样的是，这里的一行表示一个词在哪些title中出现了（一行就是之前说的一维feature），一列表示一个title中有哪些词，（这个矩阵其实是我们之前说的那种一行是一个sample的形式的一种转置，这个会使得我们的左右奇异向量的意义产生变化，但是不会影响我们计算的过程）。比如说T1这个title中就有guide、investing、market、stock四个词，各出现了一次，我们将这个矩阵进行SVD，得到下面的矩阵：

[](http://images.cnblogs.com/cnblogs_com/LeftNotEasy/201101/201101192226389491.png)      左奇异向量表示词的一些特性，右奇异向量表示文档的一些特性，中间的奇异值矩阵表示左奇异向量的一行与右奇异向量的一列的重要程序，数字越大越重要。

      继续看这个矩阵还可以发现一些有意思的东西，首先，左奇异向量的第一列表示每一个词的出现频繁程度，虽然不是线性的，但是可以认为是一个大概的描述，比如book是0.15对应文档中出现的2次，investing是0.74对应了文档中出现了9次，rich是0.36对应文档中出现了3次；

      其次，右奇异向量中一的第一行表示每一篇文档中的出现词的个数的近似，比如说，T6是0.49，出现了5个词，T2是0.22，出现了2个词。

      然后我们反过头来看，我们可以将左奇异向量和右奇异向量都取后2维（之前是3维的矩阵），投影到一个平面上，可以得到：

[](http://images.cnblogs.com/cnblogs_com/LeftNotEasy/201101/20110119222640769.png)     在图上，每一个红色的点，都表示一个词，每一个蓝色的点，都表示一篇文档，这样我们可以对这些词和文档进行聚类，比如说stock 和 market可以放在一类，因为他们老是出现在一起，real和estate可以放在一类，dads，guide这种词就看起来有点孤立了，我们就不对他们进行合并了。按这样聚类出现的效果，可以提取文档集合中的近义词，这样当用户检索文档的时候，是用语义级别（近义词集合）去检索了，而不是之前的词的级别。这样一减少我们的检索、存储量，因为这样压缩的文档集合和PCA是异曲同工的，二可以提高我们的用户体验，用户输入一个词，我们可以在这个词的近义词的集合中去找，这是传统的索引无法做到的。

     不知道按这样描述，再看看吴军老师的文章，是不是对SVD更清楚了？:-D

**参考资料：**

1）A Tutorial on Principal Component Analysis, Jonathon Shlens   
     这是我关于用SVD去做PCA的主要参考资料   
2）<http://www.ams.org/samplings/feature-column/fcarc-svd>   
     关于svd的一篇概念好文，我开头的几个图就是从这儿截取的   
3）<http://www.puffinwarellc.com/index.php/news-and-articles/articles/30-singular-value-decomposition-tutorial.html>   
     另一篇关于svd的入门好文   
4）<http://www.puffinwarellc.com/index.php/news-and-articles/articles/33-latent-semantic-analysis-tutorial.html>   
     svd与LSI的好文，我后面LSI中例子就是来自此   
5）<http://www.miislita.com/information-retrieval-tutorial/svd-lsi-tutorial-1-understanding.html>   
     另一篇svd与LSI的文章，也还是不错，深一点，也比较长   
6）Singular Value Decomposition and Principal Component Analysis, Rasmus Elsborg Madsen, Lars Kai Hansen and Ole Winther, 2004   
     跟1）里面的文章比较类似

## [奇异值分解及几何意义](http://blog.csdn.net/redline2005/article/details/24100293)

2014-04-19 11:34 20621人阅读 [评论](http://blog.csdn.net/redline2005/article/details/24100293#comments)(13) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/redline2005/article/details/24100293#report)

本文章已收录于：

http://static.blog.csdn.net/images/category_icon.jpg分类：

matrix analysis and applicatio（2） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpghttp://static.blog.csdn.net/images/arrow_triangle_up.jpg

[作者同类文章](http://blog.csdn.net/redline2005/article/category/2205859)*X*

**PS：一直以来对SVD分解似懂非懂，此文为译文，原文以细致的分析+大量的可视化图形演示了SVD的几何意义。能在有限的篇幅把这个问题讲解的如此清晰，实属不易。原文举了一个简单的图像处理问题，简单形象，真心希望路过的各路朋友能从不同的角度阐述下自己对SVD实际意义的理解，比如 个性化推荐中应用了SVD，文本以及Web挖掘的时候也经常会用到SVD。**

**英文原文：**[We recommend a singular value decomposition](http://blog.csdn.net/redline2005/article/details/24099377)

简介

SVD实际上是数学专业内容，但它现在已经渗入到不同的领域中。SVD的过程不是很好理解，因为它不够直观，但它对矩阵分解的效果却非常好。比如，Netflix（一个提供在线电影租赁的公司）曾经就悬赏100万美金，如果谁能提高它的电影推荐系统评分预测准确率提高10%的话。令人惊讶的是，这个目标充满了挑战，来自世界各地的团队运用了各种不同的技术。最终的获胜队伍"BellKor's Pragmatic Chaos"采用的核心**[算法](http://lib.csdn.net/base/datastructure" \o "算法与数据结构知识库" \t "_blank)**就是基于SVD。

SVD提供了一种非常便捷的矩阵分解方式，能够发现数据中十分有意思的潜在模式。在这篇文章中，我们将会提供对SVD几何上的理解和一些简单的应用实例。

**线性变换的几何意义**(**The geometry of linear transformations**)

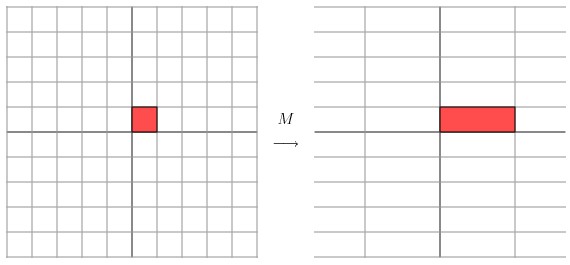
让我们来看一些简单的线性变换例子，以 2 X 2 的线性变换矩阵为例，首先来看一个较为特殊的，对角矩阵：

http://www.ams.org/featurecolumn/images/august2009/img.0.gif

从几何上讲，**M** 是将二维平面上的点(x,y)经过线性变换到另外一个点的变换矩阵，如下图所示

http://www.ams.org/featurecolumn/images/august2009/img.1.gif

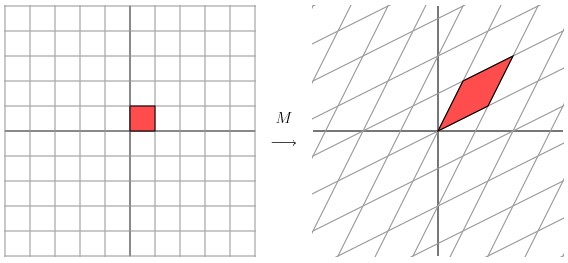
变换的效果如下图所示，变换后的平面仅仅是沿 **X** 水平方面进行了拉伸3倍，垂直方向是并没有发生变化。



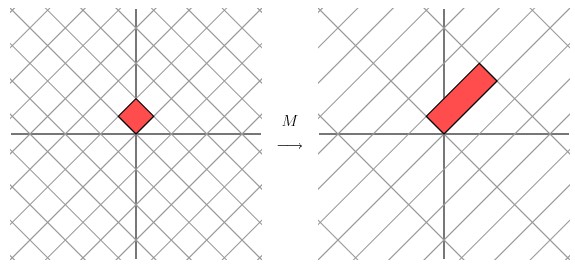
现在看下矩阵

http://www.ams.org/featurecolumn/images/august2009/img.2.gif

这个矩阵产生的变换效果如下图所示



   这种变换效果看起来非常的奇怪，在实际环境下很难描述出来变换的规律 ( 这里应该是指无法清晰辨识出旋转的角度，拉伸的倍数之类的信息)。还是基于上面的对称矩阵，假设我们把左边的平面旋转45度角，然后再进行矩阵**M** 的线性变换，效果如下图所示：



看起来是不是有点熟悉？ 对的，经过 **M** 线性变换后，跟前面的对角矩阵的功能是相同的，都是将网格沿着一个方向拉伸了3倍。

这里的 **M** 是一个特例，因为它是对称的。非特殊的就是我们在实际应用中经常遇见一些 非对称的，非方阵的矩阵。如上图所示，如果我们有一个 2 X 2 的对称矩阵**M** 的话，我们先将网格平面旋转一定的角度，**M** 的变换效果就是在两个维度上进行拉伸变换了。

用更加数学的方式进行表示的话，给定一个对称矩阵 **M** ，我们可以找到一些相互正交**Vi** ，满足**MVi** 就是沿着**Vi** 方向的拉伸变换，公式如下：

M**vi** = λi**vi**

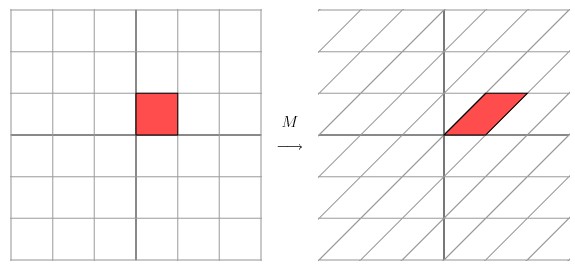
这里的 λi 是拉伸尺度(scalar)。从几何上看，**M** 对向量 **Vi** 进行了拉伸，映射变换。**Vi** 称作矩阵 M 的特征向量(eigenvector)，λi 称作为矩阵**M** 特征值(eigenvalue)。这里有一个非常重要的定理，对称矩阵**M** 的特征向量是相互正交的。

**如果我们用这些特征向量对网格平面进行线性变换的话，再通过 M 矩阵对网格平面进行线性换的效果跟对M 矩阵的特征向量进行线性变换的效果是一样的。**

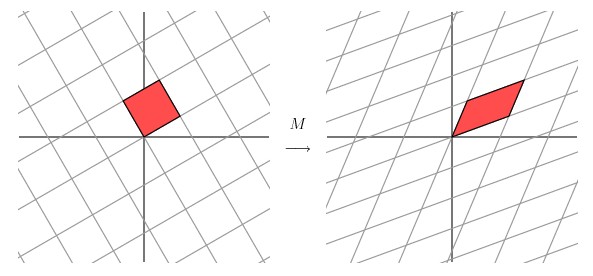
对于更为普通的矩阵而言，我们该怎么做才能让一个原来就是相互垂直的网格平面(orthogonal grid), 线性变换成另外一个网格平面同样垂直呢？**PS：**这里的垂直如图所示，就是两根交错的线条是垂直的。

http://www.ams.org/featurecolumn/images/august2009/img.3.gif

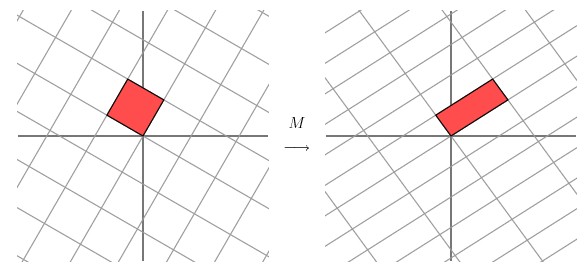
经过上述矩阵变换以后的效果如图



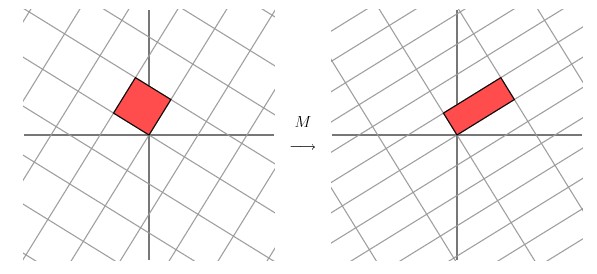
从图中可以看出，并没有达到我们想要的效果。我们把网格平面旋转 30 度角的话，然后再进行同样的线性变换以后的效果，如下图所示



让我们来看下网格平面旋转60度角的时候的效果。



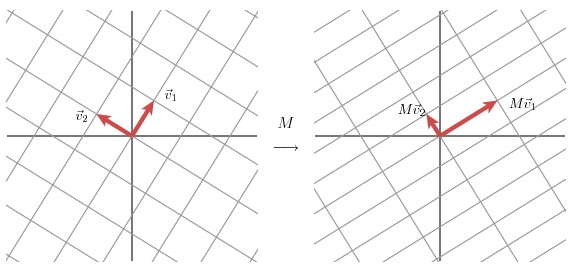
嗯嗯，这个看起来挺不错的样子。如果在精确一点的话，应该把网格平面旋转 58.28 度才能达到理想的效果。



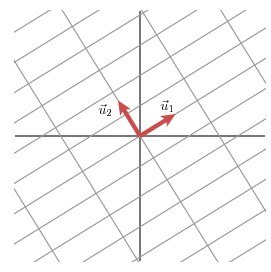
**几何意义**

该部分是从几何层面上去理解二维的SVD：对于任意的 2 x 2 矩阵，通过SVD可以将一个相互垂直的网格(orthogonal grid)变换到另外一个相互垂直的网格。

我们可以通过向量的方式来描述这个事实: 首先，选择两个相互正交的单位向量 **v1**和 **v2**, 向量M**v1** 和 M**v2** 正交。



**u1** 和**u2**分别表示M**v1** 和 M**v2**的单位向量，σ1 \* **u1** =  M**v1** 和 σ2 \* **u2** =  M**v2**。σ1 和 σ2分别表示这不同方向向量上的模，也称作为矩阵M 的奇异值。



这样我们就有了如下关系式

M**v1** = σ1**u1**   
M**v2** = σ2**u2**

我们现在可以简单描述下经过 M 线性变换后的向量 x 的表达形式。由于向量**v1** 和**v2**是正交的单位向量，我们可以得到如下式子：

**x** = (**v1**http://www.ams.org/featurecolumn/images/august2009/cdot.gif**x**)**v1** + (**v2**http://www.ams.org/featurecolumn/images/august2009/cdot.gif**x**)**v2**

这就意味着：

M**x** = (**v1**http://www.ams.org/featurecolumn/images/august2009/cdot.gif**x**)M**v1** + (**v2**http://www.ams.org/featurecolumn/images/august2009/cdot.gif**x**)M**v2**   
M**x** = (**v1**http://www.ams.org/featurecolumn/images/august2009/cdot.gif**x**) σ1**u1** + (**v2**http://www.ams.org/featurecolumn/images/august2009/cdot.gif**x**) σ2**u2**

向量内积可以用向量的转置来表示，如下所示

**v**http://www.ams.org/featurecolumn/images/august2009/cdot.gif**x** = **v**T**x**

最终的式子为

M**x** = **u1**σ1 **v1**T**x** + **u2**σ2 **v2**T**x**   
M =**u1**σ1 **v1**T + **u2**σ2 **v2**T

上述的式子经常表示成

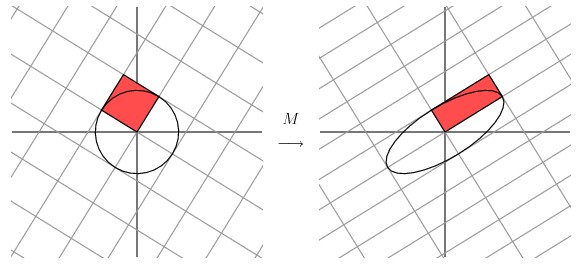
M =UΣVT

**u** 矩阵的列向量分别是**u1**,**u2**，Σ是一个对角矩阵，对角元素分别是对应的σ1 和 σ2，**V**矩阵的列向量分别是**v1**,**v2**。上角标T 表示矩阵 **V** 的转置。

   这就表明任意的矩阵 M 是可以分解成三个矩阵。**V**表示了原始域的标准正交基，**u** 表示经过M 变换后的co-domain的标准正交基，Σ表示了**V** 中的向量与**u**中相对应向量之间的关系。(V describes an orthonormal basis in the domain, and U describes an orthonormal basis in the co-domain, and Σ describes how much the vectors in V are stretched to give the vectors in U.)

**如何获得奇异值分解？**( How do we find the singular decomposition? )

   事实上我们可以找到任何矩阵的奇异值分解，那么我们是如何做到的呢？假设在原始域中有一个单位圆，如下图所示。经过 M 矩阵变换以后在co-domain中单位圆会变成一个椭圆，它的长轴(M**v1**)和短轴(M**v2**)分别对应转换后的两个标准正交向量，也是在椭圆范围内最长和最短的两个向量。



换句话说，定义在单位圆上的函数|M**x**|分别在**v1**和**v2**方向上取得最大和最小值。这样我们就把寻找矩阵的奇异值分解过程缩小到了优化函数|M**x**|上了。结果发现（具体的推到过程这里就不详细介绍了）这个函数取得最优值的向量分别是矩阵 MT M 的特征向量。由于MTM是对称矩阵，因此不同特征值对应的特征向量都是互相正交的，我们用vi 表示MTM的所有特征向量。奇异值σi = |M**vi**| ， 向量 **ui** 为 M**vi** 方向上的单位向量。但为什么**ui**也是正交的呢？

推倒如下：

σi 和 σj分别是不同两个奇异值

M**vi** = σi**ui**   
M**vj** = σj**uj**.

我们先看下M**vi**http://www.ams.org/featurecolumn/images/august2009/cdot.gifM**vj**，并假设它们分别对应的奇异值都不为零。一方面这个表达的值为0，推到如下

M**vi**http://www.ams.org/featurecolumn/images/august2009/cdot.gif M**vj** = **vi**TMT M**vj** = **vi**http://www.ams.org/featurecolumn/images/august2009/cdot.gif MTM**vj** = λj**vi**http://www.ams.org/featurecolumn/images/august2009/cdot.gif **vj** = 0

另一方面，我们有

M**vi**http://www.ams.org/featurecolumn/images/august2009/cdot.gif M**vj** = σiσj **ui**http://www.ams.org/featurecolumn/images/august2009/cdot.gif **uj** = 0

因此，**ui** 和**uj**是正交的。但实际上，这并非是求解奇异值的方法，效率会非常低。这里也主要不是讨论如何求解奇异值，为了演示方便，采用的都是二阶矩阵。

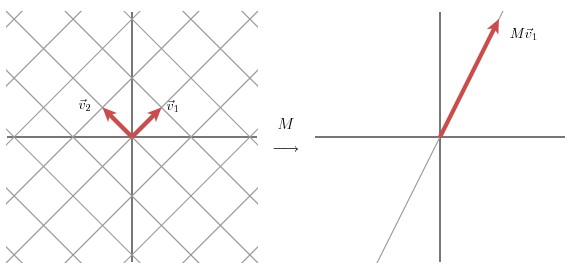
**应用实例(Another example)**

现在我们来看几个实例。

**实例一**

http://www.ams.org/featurecolumn/images/august2009/img.4.gif

经过这个矩阵变换后的效果如下图所示



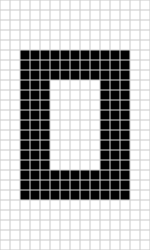
在这个例子中，第二个奇异值为 0，因此经过变换后只有一个方向上有表达。

M =**u1**σ1 **v1**T.

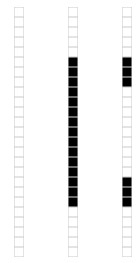
换句话说，如果某些奇异值非常小的话，其相对应的几项就可以不同出现在矩阵 M 的分解式中。因此，我们可以看到矩阵 M 的秩的大小等于非零奇异值的个数。

**实例二**

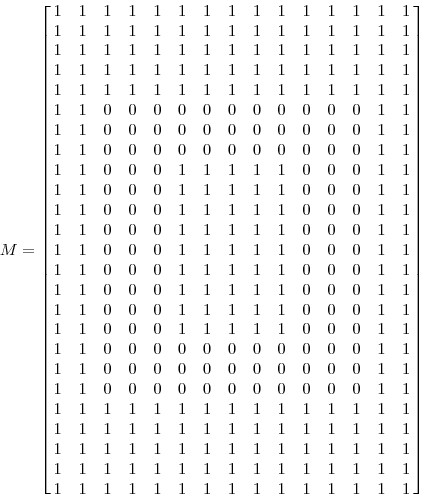
我们来看一个奇异值分解在数据表达上的应用。假设我们有如下的一张 15 x 25 的图像数据。



如图所示，该图像主要由下面三部分构成。



我们将图像表示成 15 x 25 的矩阵，矩阵的元素对应着图像的不同像素，如果像素是白色的话，就取 1，黑色的就取 0. 我们得到了一个具有375个元素的矩阵，如下图所示



如果我们对矩阵M进行奇异值分解以后，得到奇异值分别是

σ1 = 14.72   
σ2 = 5.22   
σ3 = 3.31

矩阵M就可以表示成

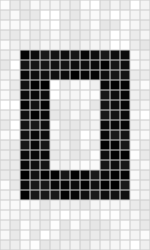
M=**u**1σ1 **v**1T + **u**2σ2 **v**2T + **u**3σ3 **v**3T

**vi**具有15个元素，**ui** 具有25个元素，σi 对应不同的奇异值。如上图所示，我们就可以用123个元素来表示具有375个元素的图像数据了。

**实例三**

**减噪(noise reduction)**

前面的例子的奇异值都不为零，或者都还算比较大，下面我们来探索一下拥有零或者非常小的奇异值的情况。通常来讲，大的奇异值对应的部分会包含更多的信息。比如，我们有一张扫描的，带有噪声的图像，如下图所示



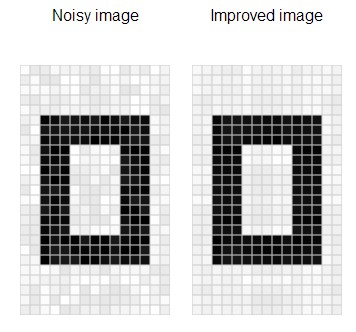
我们采用跟实例二相同的处理方式处理该扫描图像。得到图像矩阵的奇异值：

σ1 = 14.15   
σ2 = 4.67   
σ3 = 3.00   
σ4 = 0.21   
σ5 = 0.19   
...  
σ15 = 0.05

很明显，前面三个奇异值远远比后面的奇异值要大，这样矩阵 M 的分解方式就可以如下：

M http://www.ams.org/featurecolumn/images/august2009/approx.gif **u**1σ1 **v**1T + **u**2σ2 **v**2T + **u**3σ3 **v**3T

经过奇异值分解后，我们得到了一张降噪后的图像。

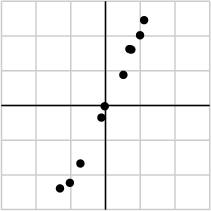


实例四

**数据分析(data analysis)**

我们搜集的数据中总是存在噪声：无论采用的设备多精密，方法有多好，总是会存在一些误差的。如果你们还记得上文提到的，大的奇异值对应了矩阵中的主要信息的话，运用SVD进行数据分析，提取其中的主要部分的话，还是相当合理的。

作为例子，假如我们搜集的数据如下所示：



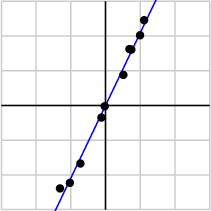
我们将数据用矩阵的形式表示：

http://image.sciencenet.cn/album/201306/14/161851gmlgg667vmvvh6gl.jpg

经过奇异值分解后，得到

σ1 = 6.04   
σ2 = 0.22

由于第一个奇异值远比第二个要大，数据中有包含一些噪声，第二个奇异值在原始矩阵分解相对应的部分可以忽略。经过SVD分解后，保留了主要样本点如图所示



就保留主要样本数据来看，该过程跟PCA( principal component analysis)技术有一些联系，PCA也使用了SVD去检测数据间依赖和冗余信息.

**总结(Summary)**

   这篇文章非常的清晰的讲解了SVD的几何意义，不仅从数学的角度，还联系了几个应用实例形象的论述了SVD是如何发现数据中主要信息的。在netflix prize中许多团队都运用了矩阵分解的技术，该技术就来源于SVD的分解思想，矩阵分解算是SVD的变形，但思想还是一致的。之前算是能够运用矩阵分解技术于个性化推荐系统中，但理解起来不够直观，阅读原文后醍醐灌顶，我想就从SVD能够发现数据中的主要信息的思路，就几个方面去思考下如何利用数据中所蕴含的潜在关系去探索个性化推荐系统。也希望路过的各位大侠不吝分享呀。

***References:***

Gilbert Strang,**Linear Algebra and Its Applications**. Brooks Cole

William H. Presset al,**Numercial Recipes in C: The Art of Scientific Computing**. Cambridge University Press.

Dan Kalman, [A Singularly Valuable Decomposition: The SVD of a Matrix,](http://www.math.umn.edu/~lerman/math5467/svd.pdf) The College Mathematics Journal **27** (1996), 2-23.

[If You Liked This, You're Sure to Love That](http://www.nytimes.com/2008/11/23/magazine/23Netflix-t.html),The New York Times, November 21, 2008.

## [SVD 详解 与 spark实战](http://blog.csdn.net/bitcarmanlee/article/details/52068118)

标签： [SVD分解](http://www.csdn.net/tag/SVD%e5%88%86%e8%a7%a3)[spark实战](http://www.csdn.net/tag/spark%e5%ae%9e%e6%88%98)[详解](http://www.csdn.net/tag/%e8%af%a6%e8%a7%a3)

2016-07-29 19:47 2983人阅读 [评论](http://blog.csdn.net/bitcarmanlee/article/details/52068118#comments)(1) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/bitcarmanlee/article/details/52068118#report)

本文章已收录于：

http://static.blog.csdn.net/images/category_icon.jpg分类：

matrix*（7）* http://static.blog.csdn.net/images/arrow_triangle%20_down.jpghttp://static.blog.csdn.net/images/arrow_triangle_up.jpg

[作者同类文章](http://blog.csdn.net/bitcarmanlee/article/category/6196173)*X*

版权声明：本文为博主原创文章，未经博主允许不得转载。

目录[(?)[+]](http://blog.csdn.net/bitcarmanlee/article/details/52068118)

[前言](http://blog.csdn.net/bitcarmanlee/article/details/52068118#t0)

[特征值特征向量特征值分解](http://blog.csdn.net/bitcarmanlee/article/details/52068118#t1)

[SVD分解](http://blog.csdn.net/bitcarmanlee/article/details/52068118#t2)

[SVD分解的应用](http://blog.csdn.net/bitcarmanlee/article/details/52068118#t3)

[降维](http://blog.csdn.net/bitcarmanlee/article/details/52068118#t4)

[压缩](http://blog.csdn.net/bitcarmanlee/article/details/52068118#t5)

[spark中SVD分解的计算方法](http://blog.csdn.net/bitcarmanlee/article/details/52068118#t6)

[spark SVD实战](http://blog.csdn.net/bitcarmanlee/article/details/52068118#t7)

**1.前言**

一般提到特征值分解(eigenvalue decomposition)或者奇异值分解(singular value decomposition)，大多数同学脑海里的第一反应就是一大堆矩阵以及数学计算方法。确实，学校学习阶段，不管是学线性代数或者矩阵分析，对于这部分内容，或者说绝大部分内容，老师一上来都是吧啦吧啦给你一堆定理推论或者公理，然后就是哗啦哗啦一堆公式出来，告诉你怎么计算。最后再讲个一两道例题，这个知识点就算讲完了。至于这些公式的来龙去脉，尤其是这些公式定理在实际中有什么用，能解决什么实际问题，老师很少有谈及。所以大家普遍反映对于线性代数矩阵分析这类课程，觉得特别枯燥。学完以后，即使考试过了会做题，随便再问几个为什么，基本也是一脸懵逼的状态。至少我当年的学习经历就是这样滴。   
等出学校以后发现，实际上在学校学的这些内容，都是非常有用而且有明确的数学或者物理意义的。在学校的时候，老师一般都会告诉你这个很有用，但是给解释清楚的，确实很少。今天，我就按照自己的理解，试图给大家将特征值分解与SVD的来龙去脉解释清楚。如果有哪里不对或者理解有偏差，还请大家海涵并指出。

**2.特征值、特征向量、特征值分解**

特征值特征向量是贯穿整个线性代数与矩阵分析的主线之一。那么特征值特征向量除了课本上公式的描述以外，到底有什么实际意义呢？   
在[http://blog.csdn.net/bitcarmanlee/article/details/52067985](http://blog.csdn.net/bitcarmanlee/article/details/52067985" \t "_blank)一文中，为大家解释了一个核心观点：矩阵是线性空间里的变换的描述。在有了这个认识的基础上，咱们接着往下。   
特征值的定义很简单：*Ax*=*λx* 。其中*A* 为矩阵,*λ* 为特征值，*x* 为特征向量。不知道大家想过没有：为什么一个向量，跟一个数相乘的效果，与跟一个矩阵的效果相乘是一样的呢？   
这得用到我们先前的结论：矩阵是线性空间里的变换的描述。矩阵*A* 与向量相乘，本质上对向量*x* 进行一次线性转换（旋转或拉伸），而该转换的效果为常数*c* 乘以向量*x* （即只进行拉伸）。当我们求特征值与特征向量的时候，就是为了求矩阵*A* 能使哪些向量（特征向量）只发生拉伸，而拉伸的程度，自然就是特征值*λ* 了。

如果还有同学没有看懂，再引用wiki百科上的一个描述：   
*N* 维非零向量 *x* 是 *N*×*N* 的矩阵 *A* 的特征向量，当且仅当下式成立：*Ax*=*λx*   
其中*λ* 为一标量，称为*x* 对应的特征值。也称*x* 为特征值*λ* 对应的特征向量。也即特征向量被施以线性变换*A* 只会使向量伸长或缩短而其方向不被改变。

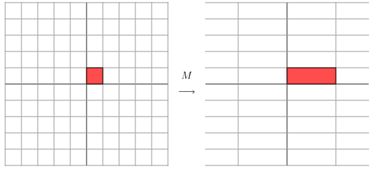
对于一个矩阵*A* ，有一组特征向量；再将这组向量进行正交化单位化，也就是我们学过的Schmidt正交化，就能得到一组正交单位向量。特征值分解，就是将矩阵*A* 分解为如下方式：

*A*=*Q*Σ*Q*−1

这其中，*Q* 是矩阵*A* 的特征向量组成的矩阵，Σ 则是一个对角阵，对角线上的元素就是特征值。

为了描述更清楚，引用网络上的一部分描述：   
对于一个矩阵*M* :

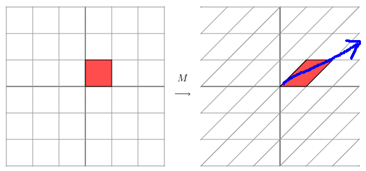
*M*=[3001]

它对应的线性变换是下面的形式：   
  
因为这个矩阵M乘以一个向量(x,y)的结果是：

[3001][*xy*]=[3*xy*]

上面的矩阵是对称的，所以这个变换是一个对x，y轴的方向一个拉伸变换（每一个对角线上的元素将会对一个维度进行拉伸变换，当值>1时，是拉长，当值<1时时缩短），当矩阵不是对称的时候，假如说矩阵是下面的样子：

[3011]

它所描述的变换是下面的样子：   


这其实是在平面上对一个轴进行的拉伸变换（如蓝色的箭头所示），在图中，蓝色的箭头是一个最主要的变化方向（变化方向可能有不止一个），如果我们想要描述好一个变换，那我们就描述好这个变换主要的变化方向就好了。反过头来看看之前特征值分解的式子，分解得到的Σ矩阵是一个对角阵，里面的特征值是由大到小排列的，这些特征值所对应的特征向量就是描述这个矩阵变化方向（从主要的变化到次要的变化排列）

当矩阵是高维的情况下，那么这个矩阵就是高维空间下的一个线性变换，这个线性变化可能没法通过图片来表示，但是可以想象，这个变换也同样有很多的变换方向，我们通过特征值分解得到的前N个特征向量，那么就对应了这个矩阵最主要的N个变化方向。我们利用这前N个变化方向，就可以近似这个矩阵（变换）。也就是之前说的：提取这个矩阵最重要的特征。总结一下，特征值分解可以得到特征值与特征向量，特征值表示的是这个特征到底有多重要，而特征向量表示这个特征是什么，可以将每一个特征向量理解为一个线性的子空间，我们可以利用这些线性的子空间干很多的事情。不过，特征值分解也有很多的局限，比如说变换的矩阵必须是方阵。

以上的图片及内容来自LeftNotEasy的博客内容。感觉描述还是比较到位。

**3.SVD分解**

前面啰啰嗦嗦说了这么多基础，终于轮到咱们的主角：SVD登场了。   
首先我们来看看奇异值的定义：对于一个矩阵*A* ，有：(*ATA*)*ν*=*λν*

那么向量*x* 就是*A* 的右奇异向量。并且：   
奇异值：*σi*=*λi*−−√   
左奇异向量：*μi*=1*σiAνi*

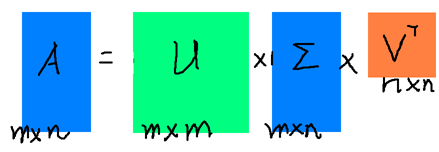
咱们前面讲了那么多的特征值与特征值分解，而且特征值分解是一个提取矩阵特征很不错的方法。但是，特征值分解最大的问题是只能针对方阵，即*n*∗*n* 的矩阵。而在实际应用场景中，大部分不是这种矩阵。举个最简单的例子，关系型**[数据库](http://lib.csdn.net/base/mysql" \o "MySQL知识库" \t "_blank)**中的某一张表的数据存储结构就类似于一个二维矩阵，假设这个表有*m* 行，有*n* 个字段，那么这个表数据矩阵的规模就是*m*∗*n* 。很明显，在绝大部分情况下，*m* 与*n* 并不相等。如果对这个矩阵要进行特征提取，特征值分解的方法显然就行不通了。那么此时，就是SVD分解发挥威力的时候。

假设*A* 是一个*m*∗*n* 阶矩阵，如此则存在一个分解使得

*A*=*U*Σ*VT*

其中*U* 是*m*×*m* 阶酉矩阵；Σ 是*m*×*n* 阶非负实数对角矩阵；而*VT* ，即*V* 的共轭转置，是*n*×*n* 阶酉矩阵。这样的分解就称作*M* 的奇异值分解。Σ 对角线上的元素Σ*i* ,*i* 即为*M* 的奇异值。而且一般来说，我们会将Σ 上的值按从大到小的顺序排列。

通过上面对SVD的简单描述，不难发现，SVD解决了特征值分解中只能针对方阵而没法对更一般矩阵进行分解的问题。所以在实际中，SVD的应用场景比特征值分解更为通用与广泛。

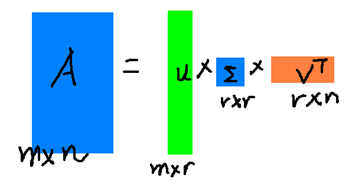
将将上面的SVD分解用一个图形象表示如下   


其中各矩阵的规模已经在上面描述过了。

截止到这里为止，很多同学会有疑问了：你这不吃饱了撑得。好好的一个矩阵*A* ，你这为毛要将他表示成三个矩阵。这三个矩阵的规模，一点也不比原来矩阵的规模小好么。而且还要做两次矩阵的乘法。要知道，矩阵乘法可是个复杂度为*O*(*n*3) 的运算。

同志们别急，请接着往下看。   
如果按照之前那种方式分解，肯定是没有任何好处的。矩阵规模大了，还要做乘法运算。关键是奇异值有个牛逼的性质：在大部分情况下，当我们把矩阵Σ 里的奇异值按从大到小的顺序呢排列以后，很容易就会发现，奇异值*σ* 减小的速度特别快。在很多时候，前10%甚至前1%的奇异值的和就占了全部奇异值和的99%以上。换句话说，大部分奇异值都很小，基本没什么卵用。。。既然这样，那我们就可以用前面r个奇异值来对这个矩阵做近似。于是，SVD也可以这么写：

*Am*×*n*≈*Um*×*r*Σ*r*×*rVr*×*n*

其中，*r*≪*m* ，*r*≪*n* 。如果用另外一幅图描述这个过程，如下图：   


看了上面这幅图，同学们是不是就恍然大悟：原来的那个大矩阵*A* ，原来可以用右边的那三个小矩阵来表示。当然如果*r* 越大，跟原来的矩阵相似度就越高。如果*r*=*n* ，那得到的就是原来的矩阵*A* 。但是这样存储与计算的成本就越高。所以，实际在使用SVD的时候，需要我们根据不同的业务场景与需求还有资源情况，合理选择*r* 的大小。本质而言，就是在计算精度与空间时间成本之间做个折中。

**4.SVD分解的应用**

SVD在实际中应用非常广泛，每个应用场景再单写一篇文章都没有问题。这里我们先不做过多的展开，先举两个最重要的方面。为了方便后面的描述，先把SVD的近似表达式再拎出来

*Am*×*n*≈*Um*×*r*Σ*r*×*rVr*×*n*

**1.降维**

通过上面的式子很容易看出，原来矩阵*A* 的特征有*n* 维。而经过SVD分解之后，完全可以用前*r* 个非零奇异值对应的奇异向量表示矩阵*A* 的主要特征。这样，就天然起到了降维的作用。

**2.压缩**

还是看上面的式子，再结合第三部分的图，也很容易看出，经过SVD分解以后，要表示原来的大矩阵*A* ，我们只需要存*U* ，Σ ，*V* 三个较小的矩阵的即可。而这三个较小矩阵的规模，加起来也远远小于原有矩阵*A* 。这样，就天然起到了压缩的作用。

**5.spark中SVD分解的计算方法**

因为SVD是如此的基础与重要，所以在任何一个**[机器学习](http://lib.csdn.net/base/machinelearning" \o "机器学习知识库" \t "_blank)**的库里，都实现了SVD的相关**[算法](http://lib.csdn.net/base/datastructure" \o "算法与数据结构知识库" \t "_blank)**，[**Spark**](http://lib.csdn.net/base/spark)里面自然也不例外。   
spark里SVD是在MLlib包里的Dimensionality Reduction里(spark版本1.6，以下所有api与代码都是基于此版本)。文档里有SVD原理的简单描述。原理前面我们已经讲过，就不在重复了。重点看看里面的Performance:   
We assume *n* is smaller than *m* .The singular values and the right singular vectors are derived from the eigenvalues and the eigenvectors of the Gramian matrix *ATA* .The matrix storing the left singular vectors *U* ,is computed via matrix multiplication as *U*=*A*(*VS*−1) , if requested by the user via the computeU parameter. The actual method to use is determined automatically based on the computational cost:

If *n* is small (*n* <100) or *k* is large compared with *n* (*k*>*n*/2 ),we compute the Gramian matrix first and then compute its top eigenvalues and eigenvectors locally on the driver. This requires a single pass with *O*(*n*2) storage on each executor and on the driver, and *O*(*n*2*k*) time on the driver.

给大家翻译一下：假设*n*<*m* 。奇异值与右奇异向量通过计算格莱姆矩阵*ATA* 的特征值特征向量可以得知。而存储左奇异向量的矩阵*U* ，是通过矩阵乘法运算*U*=*A*(*VS*−1) 计算得出的。实际计算中，是根据计算的复杂程度自动决定的：   
1.如果*n* 很小（*n*<100 ），或者*k* 与*n* 相比比较大（*k*>*n*/2 ），那么先计算格莱姆矩阵*ATA* ，再在spark的driver本地上计算其特征值与特征向量。这种方法需要在每个executor上*O*(*n*2) 的存储空间，driver上*O*(*n*2) 的存储空间，以及*O*(*n*2*k*) 的时间复杂度。   
2.如果不是上面的第一种情况，那么先用分布式的方法计算(*ATA*)*ν* 在把结果传到ARPACK上用于后续再每个driver节点上计算*ATA* 的前几个特征值与特征向量。这种方法需要*O*(*k*) 的网络传输，每个executor上*O*(*n*) 的存储，以及driver上的*O*(*nk*) 的存储。

**6.spark SVD实战**

前头这么多内容讲的都是理论。显然纯理论不是我们的风格， talk is cheap,show me the code。理论说得再美好，是骡子是马，总得拉出来遛遛。只有亲自看这代码run起来，心里才能踏实。   
关于spark的安装，环境配置等问题，同学们参考之前的博客   
<http://blog.csdn.net/bitcarmanlee/article/details/51967323> spark集群搭建   
<http://blog.csdn.net/bitcarmanlee/article/details/52048976> spark本地开发环境搭建

然后直接上源码

import org.apache.spark.{SparkConf, SparkContext}

import org.apache.spark.mllib.linalg.{Matrix, SingularValueDecomposition, Vectors, Vector}

import org.apache.spark.mllib.linalg.distributed.RowMatrix

/\*\*

\* Created by lei.wang on 16/7/29.

\*/

object SvdTest {

def main(args: Array[String]): Unit = {

val conf = new SparkConf().setAppName("SVD").setMaster("spark://your host:7077").setJars(List("your .jar file"))

val sc = new SparkContext(conf)

val data = Array(

Vectors.sparse(5,Seq((1,1.0),(3,7.0))),

Vectors.dense(2.0,0.0,3.0,4.0,5.0),

Vectors.dense(4.0,0.0,0.0,6.0,7.0))

val dataRDD = sc.parallelize(data,2)

val mat:RowMatrix = new RowMatrix(dataRDD)

val svd: SingularValueDecomposition[RowMatrix,Matrix] = mat.computeSVD(5,computeU = true)

val U:RowMatrix = svd.U //U矩阵

val s:Vector = svd.s //奇异值

val V:Matrix = svd.V //V矩阵

println(s)

}

}

这里头我们通过调用API将*r* 的值设为5，最后的输出结果中，奇异值就有5个：

...

The singular values is:

[13.029275535600473,5.368578733451684,2.5330498218813755,6.323166049206486E-8,2.0226934557075942E-8]

...

从结果很容易看出来，第一个奇异值最大，而且占了总和的将近70%。最后两个奇异值则很小，基本可以忽略不计。