教程 | 从特征分解到协方差矩阵：详细剖析和实现PCA算法

2017-07-05 机器之心

选自deeplearning4j

机器之心编译

参与：蒋思源

本文先简要明了地介绍了特征向量和其与矩阵的关系，然后再以其为基础解释协方差矩阵和主成分分析法的基本概念，最后我们结合协方差矩阵和主成分分析法实现数据降维。本文不仅仅是从理论上阐述各种重要概念，同时最后还一步步使用 Python 实现数据降维。

首先本文的特征向量是数学概念上的特征向量，并不是指由输入特征值所组成的向量。数学上，线性变换的特征向量是一个非简并的向量，其方向在该变换下不变。该向量在此变换下缩放的比例称为特征值。一个线性变换通常可以由其特征值和特征向量完全描述。如果我们将矩阵看作物理运动，那么最重要的就是运动方向（特征向量）和速度（特征值）。因为物理运动只需要方向和速度就可以描述，同理矩阵也可以仅使用特征向量和特征值描述。

其实在线性代数中，矩阵就是一个由各种标量或变量构成的表格，它和 Excel 表格并没有什么本质上的区别。只不过数学上定义了一些矩阵间的运算，矩阵运算的法则和实际内部的值并没有什么关系，只不过定义了在运算时哪些位置需要进行哪些操作。因为矩阵相当于定义了一系列运算法则的表格，那么其实它就相当于一个变换，这个变换（物理运动）可以由特征向量（方向）和特征值（速度）完全描述出来。

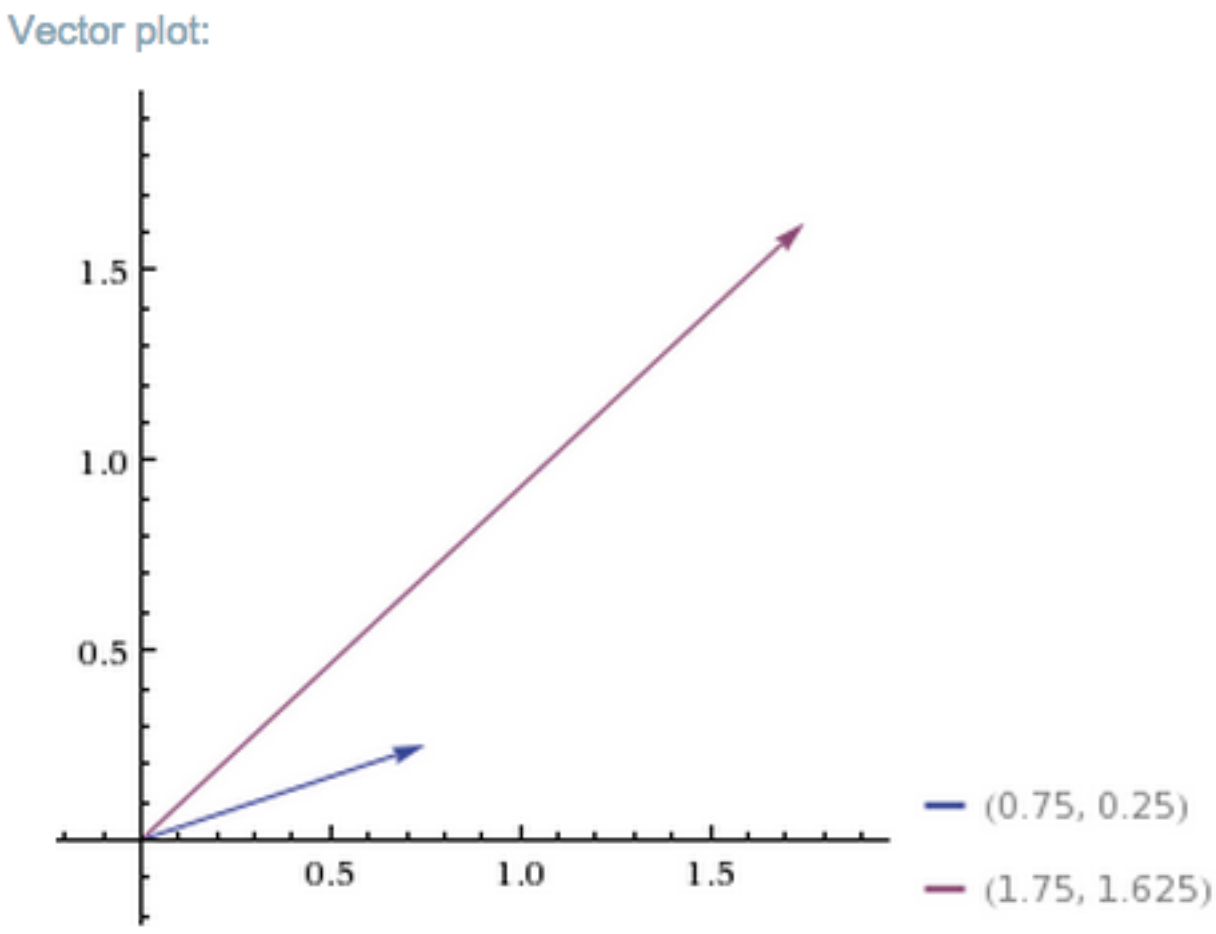
线性变换

在解释线性变换前，我们需要先了解矩阵运算到底是什么。因为我们可以对矩阵中的值统一进行如加法或乘法等运算，所以矩阵是十分高效和有用的。如下所示，如果我们将向量 v 左乘矩阵 A，我们就会得到新的向量 b，也即可以表述说矩阵 A 对输入向量 v 执行了一次线性变换，且线性变换结果为 b。因此矩阵运算 Av = b 就代表向量 v 通过一个变换（矩阵 A）得到向量 b。下面的实例展示了矩阵乘法（该类型的乘法称之为点积）是怎样进行

的：

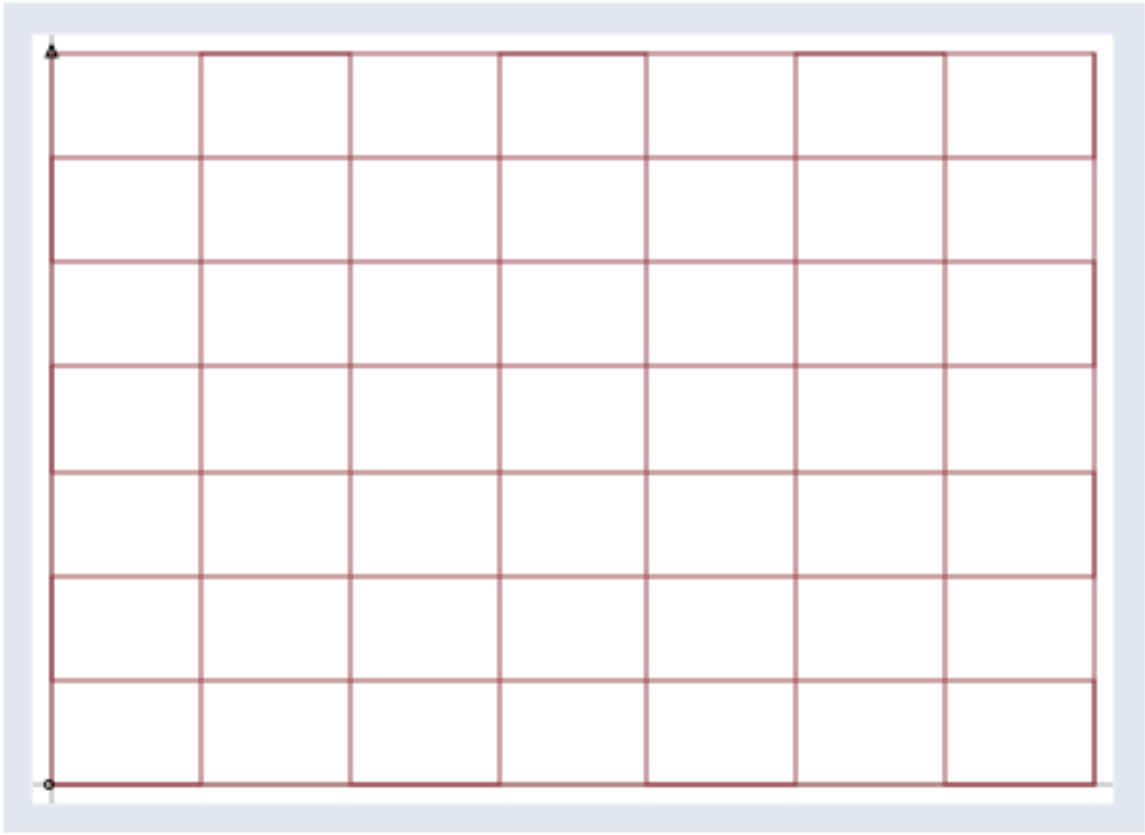
$$\underset{A}{\begin{bmatrix} 2 & 1 \\ 1.5 & 2 \end{bmatrix}} * \underset{v}{\begin{matrix} 0.75 \\ 0.25 \end{matrix}} = \underset{b}{\begin{matrix} 1.75 \\ 1.625 \end{matrix}}$$

所以矩阵 A 将向量 v 变换为向量 b。下图展示了矩阵 A 如何将更短更低的向量 v 映射到更长更高的向量 b：

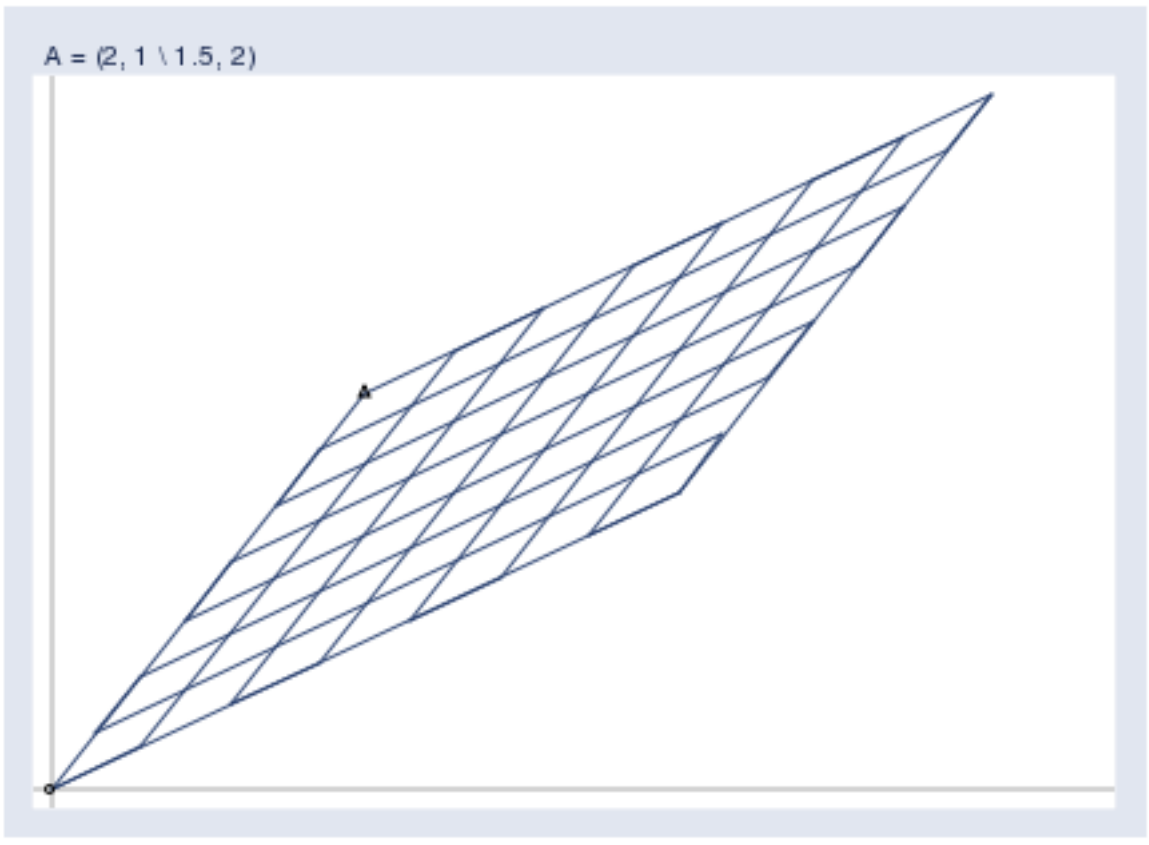


我们可以馈送其他正向量到矩阵 A 中，每一个馈送的向量都会投影到新的空间中且向右边变得更高更长。

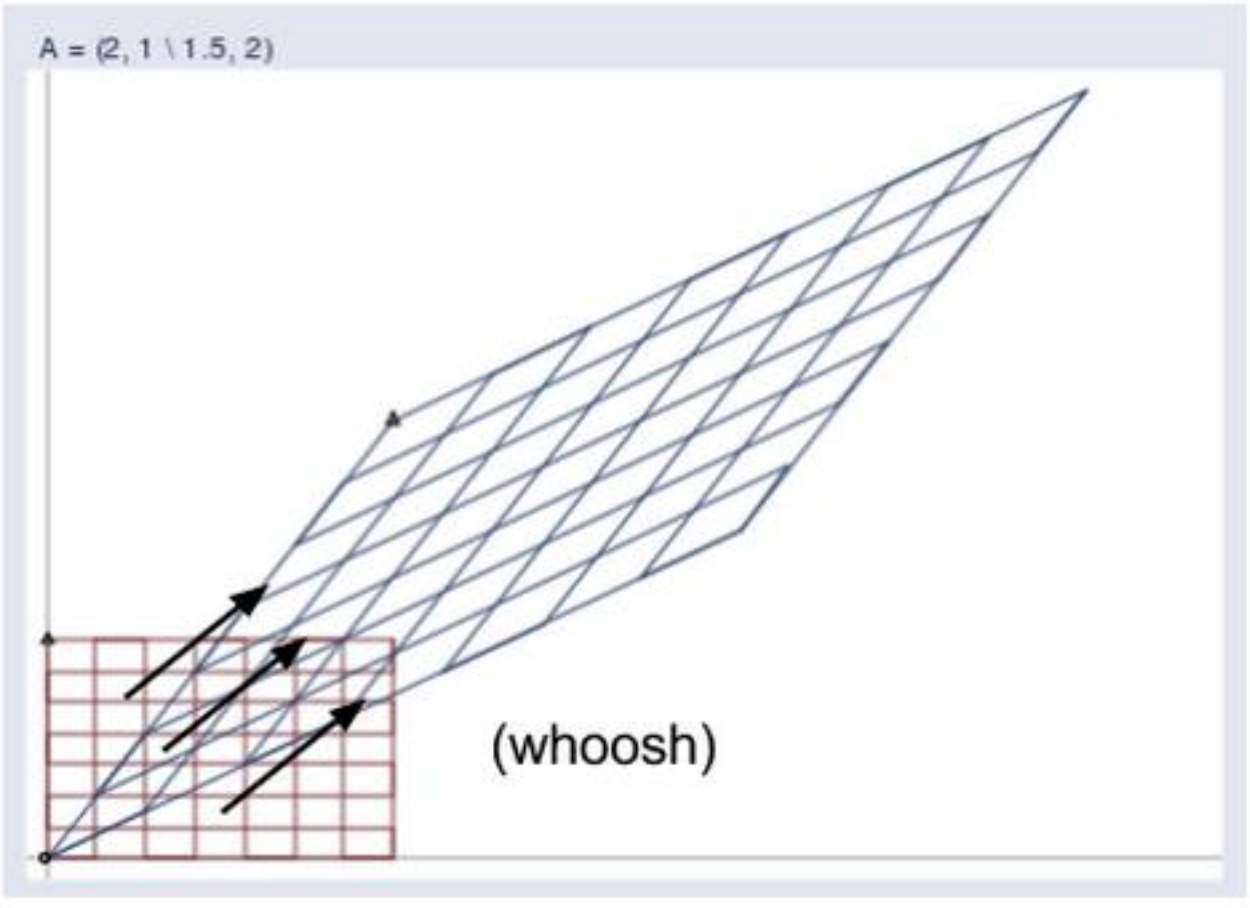
假定所有的输入向量 V 可以排列为一个标准表格，即：



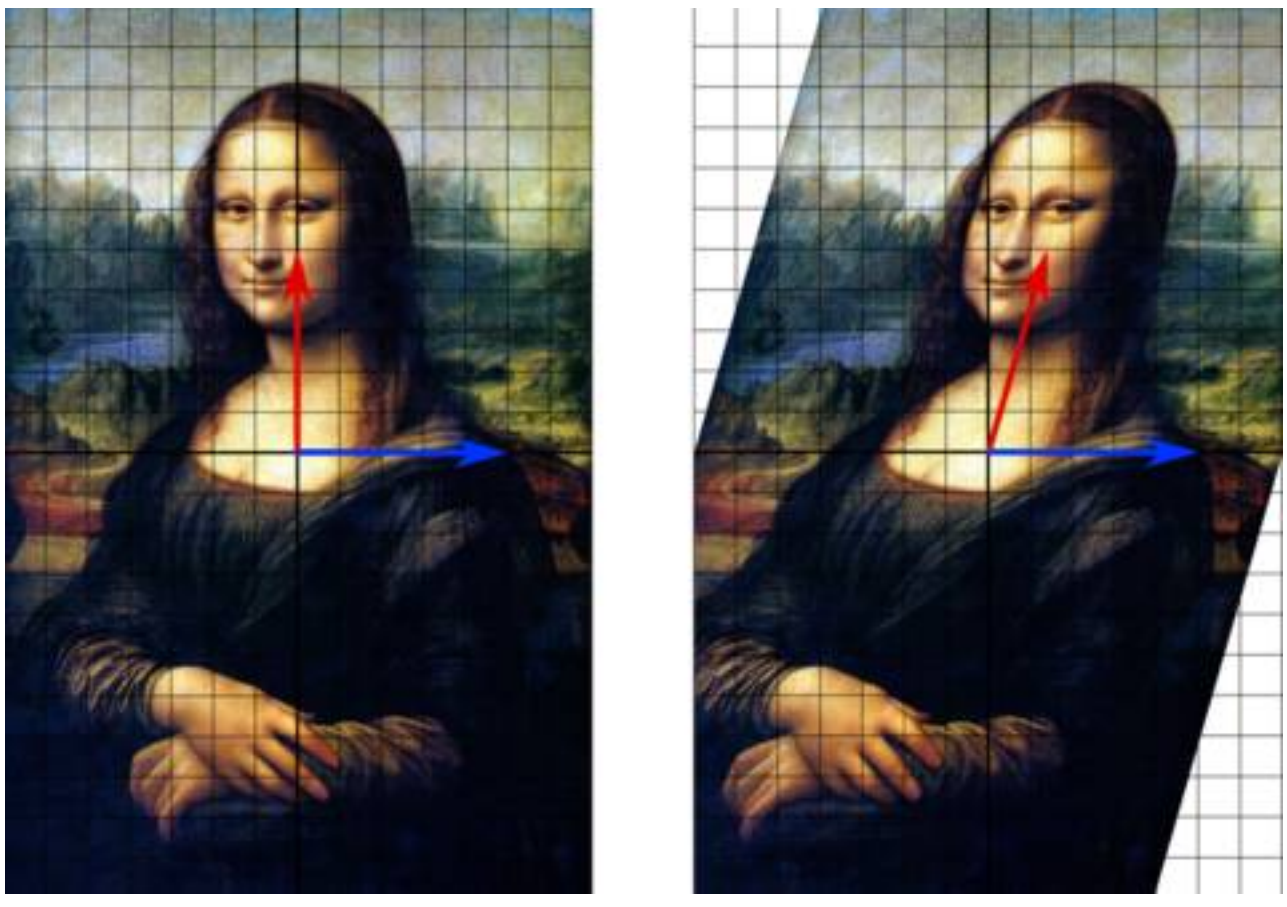
而矩阵可以将所有的输入向量 V 投影为如下所示的新空间，也即所有输出向量组成的 B：



下图可以看到输入向量空间和输出向量空间的关系，



如果假设矩阵就是一阵风，它通过有形的力量得出可见的结果。而这一阵风所吹向的方向就是特征向量，因此特征向量就表明矩阵所要变换的方向。



如上图所示，特征向量并不会改变方向，它已经指向了矩阵想要将所有输入向量都推向的方向。因此，特征向量的数学定义为：存在非零矩阵 A 和标量λ，若有向量 x 且满足以下关系式，那么 x 就为特征向量、λ为特征值。

$$Ax = \lambda x$$

特征向量同样是线性变换的不变轴，所有输入向量沿着这条轴压缩或延伸。线性变换中的线性正是表明了这种沿直线轴进行变换的特性，一般来说几阶方阵就有几个特征向量，如 3*3 矩阵有 3 个特征向量，n 阶方阵有 n 个特征向量，每一个特征向量表征一个维度上的线性变换方向。

因为特征向量提取出了矩阵变换的主要信息，因此它在矩阵分解中十分重要，即沿着特征向量对角化矩阵。因为这些特征向量表征着矩阵的重要特性，所以它们可以执行与深度神经网络中自编码器相类似的任务。引用 Yoshua Bengio 的话来说：

许多数学对象可以通过分解为更基础的组成部分而有更好的理解，因为我们会发现它们的一些广泛性属性，而不是我们选择表征它们的特性。例如整数可以分解为质因数，虽然我们表征整数的方式会因为采用二进制还是十进制而改变，但整数总可以由几个质因数表示（如 12=2 × 2 × 3），因此这种分解的性质正好是我们所需要的稳定性质。

我们可以分解一个整数为质因数而得到其自然属性，同样我们也可以分解矩阵以得到它的功能性属性，并且这种属性信息在矩阵表示为多组元素的阵列下是不明显的。矩阵分解最常见的是特征分解（eigen-decomposition），即我们将矩阵分解为一系列的特征向量和特征值。

主成分分析（PCA）

PCA 是一种寻找高维数据（图像等）模式的工具。机器学习实践上经常使用 PCA 对输入神经网络的数据进行预处理。通过聚集、旋转和缩放数据，PCA 算法可以去除一些低方差的维度而达到降维的效果，这样操作能提升神经网络的收敛速度和整体效果。

为了进一步了解 PCA 算法，我们还需要定义一些基本的统计学概念，即均值、标准差、方差和协方差。

样本均值可简单的表示为所有样本 X 的平均值，如下所示样本均值表示为：

$$\bar{X} = \frac{\sum_{i=1}^{n} X_i}{n}$$

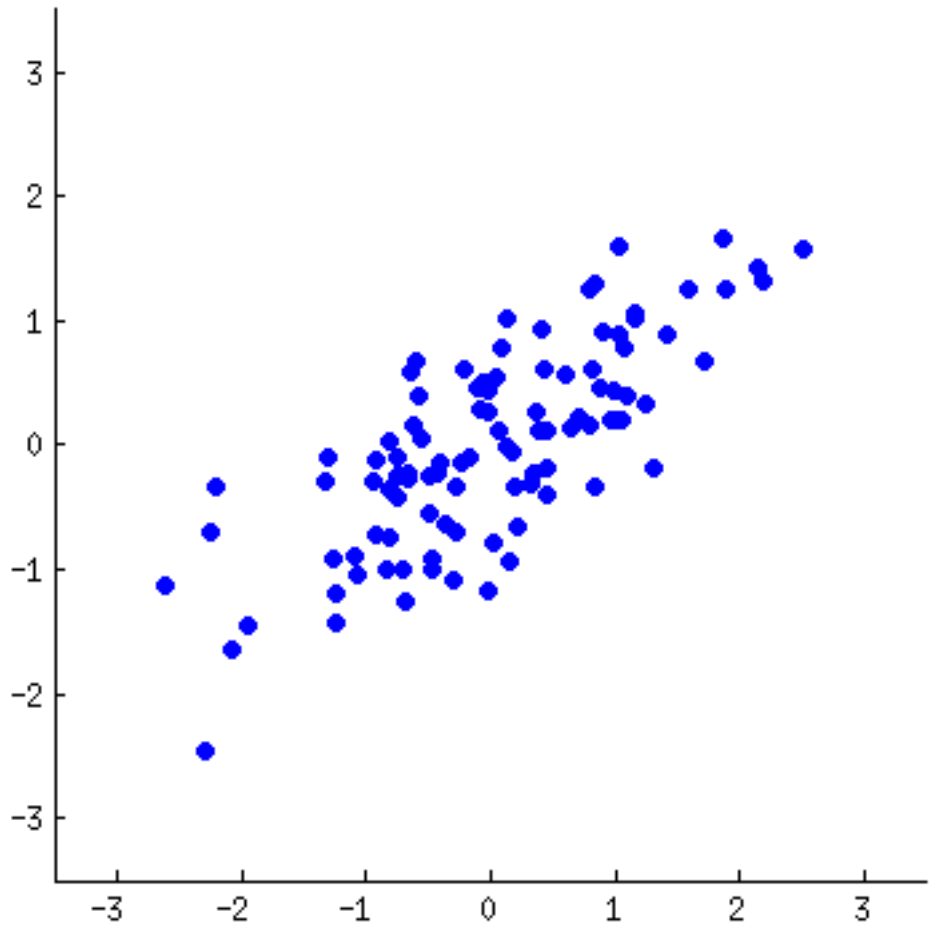
样本标准差即样本方差的平方根。即每一样本点到样本均值之间的平均距离。n 个样本的方差却只除以 n-1 是因

为样本只是真实分布的估计量，样本方差也只是真实方差的估计量。在大学课本概率论和数理统计中有证明，如果除以 n（2 阶中心矩），那么样本方差是真实方差的一致性估计，但并不是无偏估计，也就是样本方差存在系统偏差。因此我们需要对 2 阶中心矩进行调整以消除系统偏差。如下所示，样本的标准差 s 和方差 var(X) 都是无偏估计：

$$s = \sqrt{\frac{\sum_{i=1}^{n}(X_i - \bar{X})^2}{(n-1)}}$$

$$var(X) = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(X_i - \bar{X})}{(n-1)}$$

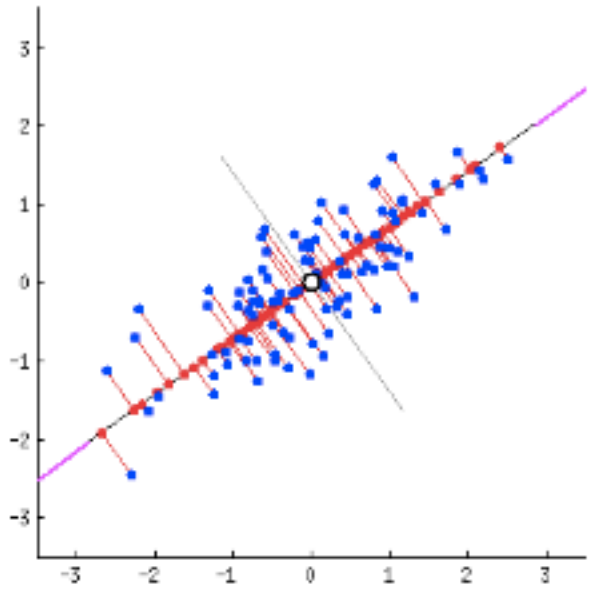
因为样本标准差和方差都是先求距离的平方再求平方根，因此距离一定是正数且不会抵消。假设我们有如下数据点（散点图）：



PCA 如线性回归那样会尝试构建一条可解释性的直线贯穿所有数据点。每一条直线表示一个「主成分」或表示自变量和因变量间的关系。数据的维度数就是主成分的数量，也即每一个数据点的特征维度。PCA 的作用就是分析这些特征，并选出最重要的特征。PCA 本质上是将方差最大的方向作为主要特征，并且在各个正交方向上将数据「去相关」，也就是让它们在不同正交方向上没有相关性。通常我们认为信息具有较大的方差，而噪声有较小的方差，信噪比就是信号与噪声的方差比，所以我们希望投影后的数据方差越大越好。因此我们认为，最好的 k 维

特征是将 n 维样本点转换为 k 维后，每一维上的样本方差都很大。

如下图所示，第一个主成分以直线（红色）的形式将散点图分为两边，并且它是保留了最大方差的。因为投影到这条直线（红色）上数据点离均值（空心点）有最大的方差，即所有蓝点到灰色线的平均距离为最大方差，所以这一个主成分将保留最多的信息。



如上所示，假设第二个主成分为垂直于红线（第一个主成分）的灰色线。当数据点投影到第二个主成分上时，它们离样本均值（空心点）的方差却非常小，即数据点到红色线的平均距离。所以红色线是最优的主成分。

协方差矩阵

前面我们已经了解矩阵其实就是一种将某个向量变换为另一个的方法，另外我们也可以将矩阵看作作用于所有数据并朝向某个方向的力。同时我们还知道了变量间的相关性可以由方差和协方差表达，并且我们希望保留最大方差以实现最优的降维。因此我们希望能将方差和协方差统一表示，并且两者均可以表示为内积的形式，而内积又与矩阵乘法密切相关。因此我们可以采用矩阵乘法的形式表示。若输入矩阵 X 有两个特征 a 和 b，且共有 m 个样本，那么有：

$$X = \begin{pmatrix} a_1 & a_2 & \cdots & a_m \\ b_1 & b_2 & \cdots & b_m \end{pmatrix}$$

如果我们用 X 左乘 X 的转置，那么就可以得出协方差矩阵：

$$\frac{1}{m}XX^{\intercal} = \begin{pmatrix} \frac{1}{m}\sum_{i=1}^{m} a_i^2 & \frac{1}{m}\sum_{i=1}^{m} a_i b_i \\ \frac{1}{m}\sum_{i=1}^{m} a_i b_i & \frac{1}{m}\sum_{i=1}^{m} b_i^2 \end{pmatrix}$$

这个矩阵对角线上的两个元素分别是两特征的方差，而其它元素是 a 和 b 的协方差。两者被统一到了一个矩阵的，因此我们可以利用协方差矩阵描述数据点之间的方差和协方差，即经验性地描述我们观察到的数据。

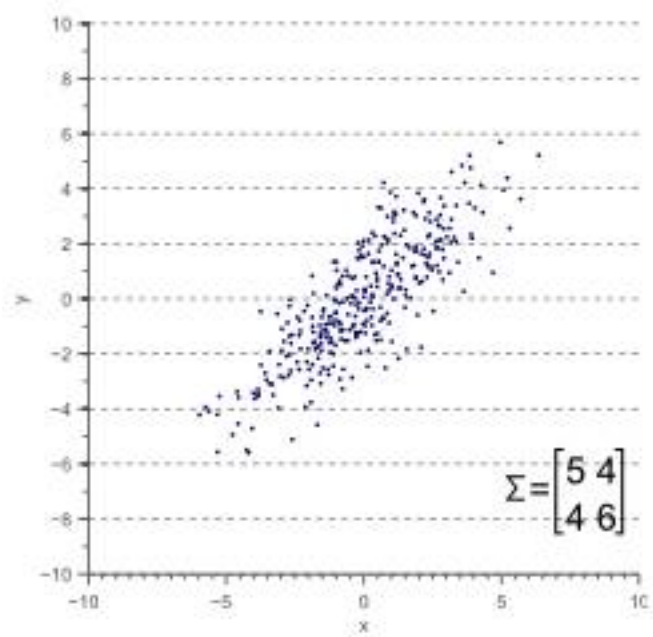
寻找协方差矩阵的特征向量和特征值就等价于拟合一条能保留最大方差的直线或主成分。因为特征向量追踪到了主成分的方向，而最大方差和协方差的轴线表明了数据最容易改变的方向。根据上述推导，我们发现达到优化目标就等价于将协方差矩阵对角化：即除对角线外的其它元素化为 0，并且在对角线上将特征值按大小从上到下排列。协方差矩阵作为实对称矩阵，其主要性质之一就是可以正交对角化，因此就一定可以分解为特征向量和特征值。

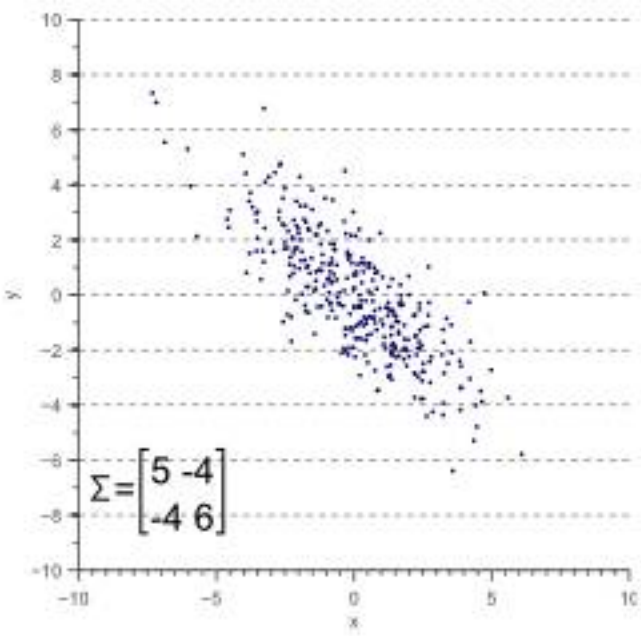
当协方差矩阵分解为特征向量和特征值之后，特征向量表示着变换方向，而特征值表示着伸缩尺度。在本例中，特征值描述着数据间的协方差。我们可以按照特征值的大小降序排列特征向量，如此我们就按照重要性的次序得到了主成分排列。

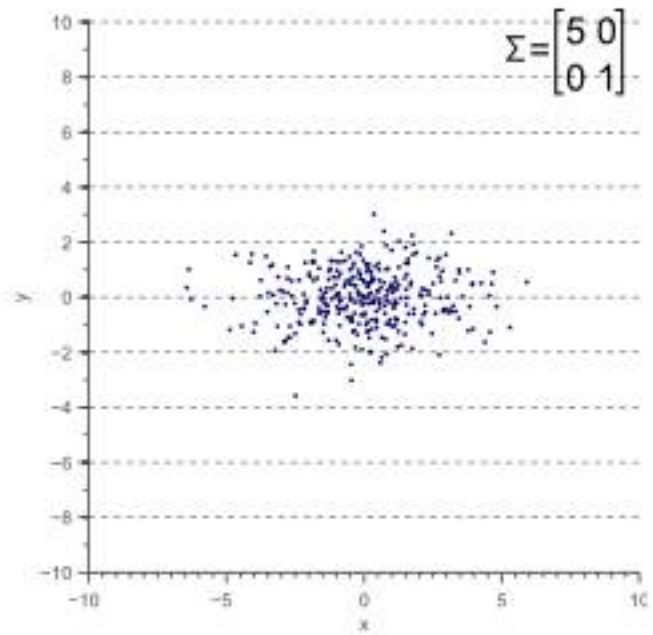
对于 2 阶方阵，一个协方差矩阵可能如下所示：

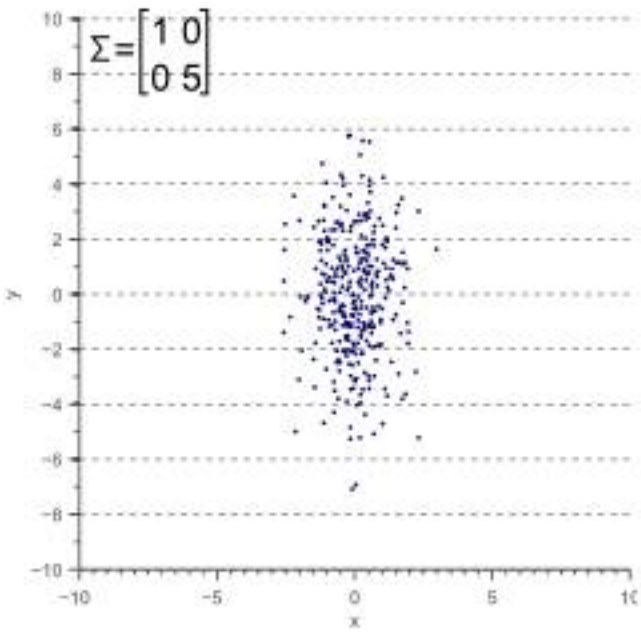
$$\begin{bmatrix} 1.07 & 0.63 \\ 0.63 & 0.64 \end{bmatrix}$$

在上面的协方差矩阵中，1.07 和 0.64 分别代表变量 x 和变量 y 的方差，而副对角线上的 0.63 代表着变量 x 和 y 之间的协方差。因为协方差矩阵为实对称矩阵（即 Aij=Aji），所以其必定可以通过正交化相似对角化。因为这两个变量的协方差为正值，所以这两个变量的分布成正相关性。如下图所示，如果协方差为负值，那么变量间就成负相关性。









注意如果变量间的协方差为零，那么变量是没有相关性的，并且也没有线性关系。因此，如果两个变量的协方差越大，相关性越大，投影到主成分后的损失就越小。我们同时可以考虑协方差和方差的计算式而了解他们的关系：

$$cov(X,Y) = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{(n-1)}$$

vs.

$$var(X) = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(X_i - \bar{X})}{(n-1)}$$

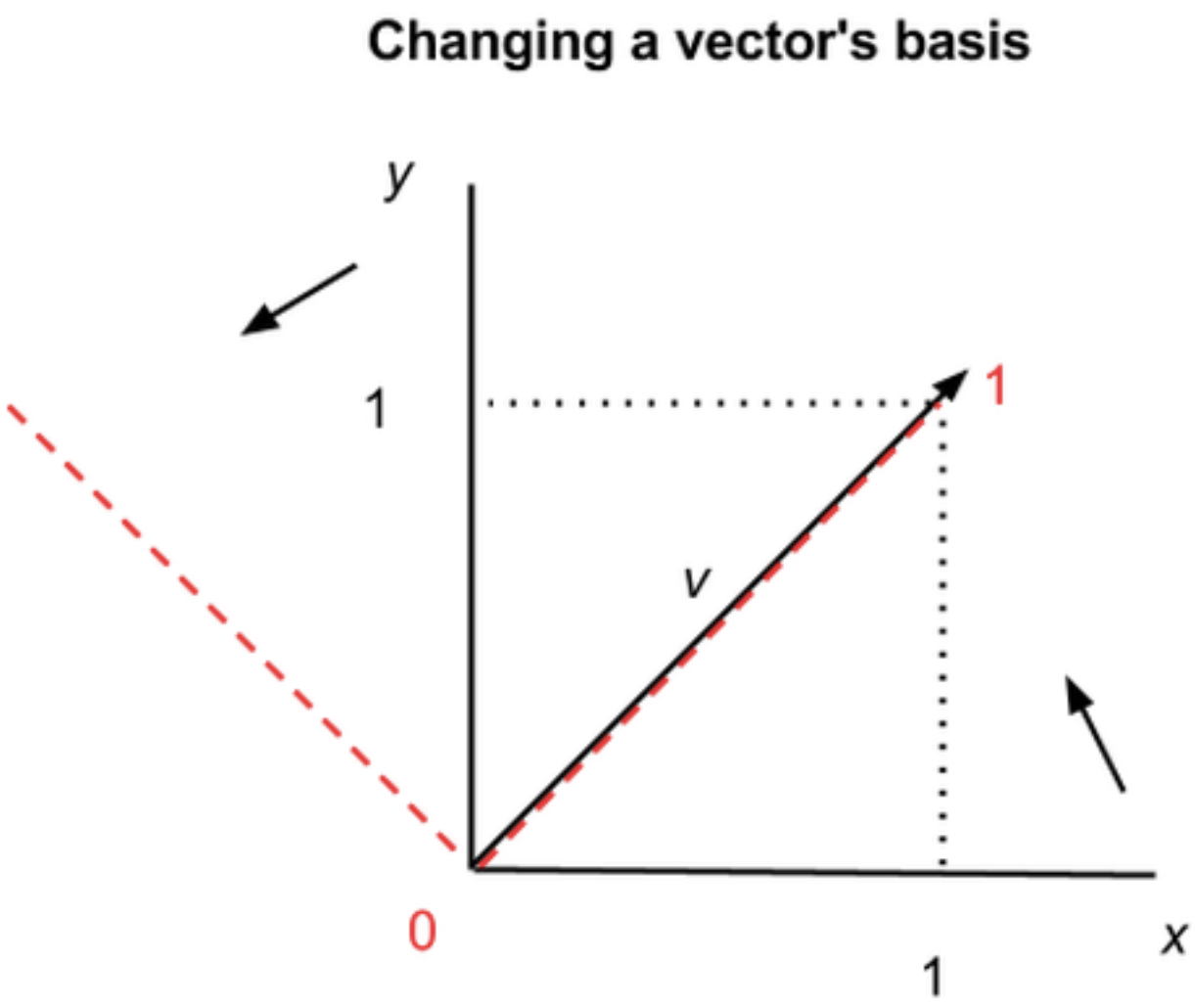
计算协方差的优处在于我们可以通过协方差的正值、负值或零值考察两个变量在高维空间中相互关系。总的来说，协方差矩阵定义了数据的形状，协方差决定了沿对角线对称分布的强度，而方差决定了沿 x 轴或 y 轴对称分

布的趋势。

基变换

因为协方差矩阵的特征向量都是彼此正交的，所以变换就相当于将 x 轴和 y 轴两个基轴换为主成分一个基轴。也就是将数据集的坐标系重新变换为由主成分作为基轴的新空间，当然这些主成分都保留了最大的方差。

我们上面所述的 x 轴和 y 轴称之为矩阵的基，即矩阵所有的值都是在这两个基上度量而来的。但矩阵的基是可以改变的，通常一组特征向量就可以组成该矩阵一组不同的基坐标，原矩阵的元素可以在这一组新的基中表达。



在上图中，我们展示了相同向量 v 如何在不同的坐标系中有不同的表达。黑色实线代表 x-y 轴坐标系而红色虚线是另外一个坐标系。在第一个坐标系中 v = (1,1)，而在第二个坐标系中 v = (1,0)。因此矩阵和向量可以在不同坐标系中等价变换。在数学上，n 维空间并没有唯一的描述，所以等价转换矩阵的基也许可以令问题更容易解决。

最后我们简单地总结一下 PCA 算法的基本概念和步骤：

首先我们得理解矩阵就相当于一个变换，变换的方向为特征向量，变换的尺度为特征值。PCA 的本质是将方差最大的方向作为主要特征，并且在各个正交方向上将数据「去相关」，即让它们在不同正交方向上没有相关性。所以我们希望将最相关的特征投影到一个主成分上而达到降维的效果，投影的标准是保留最大方差。而在实际操作中，我们希望计算特征之间的协方差矩阵，并通过对协方差矩阵的特征分解而得出特征向量和特征值。如果我们将特征值由大到小排列，相对应的特征向量所组成的矩阵就是我们所需降维后的数据。下面我们将一步步实现 PCA 算法。

输入数据：

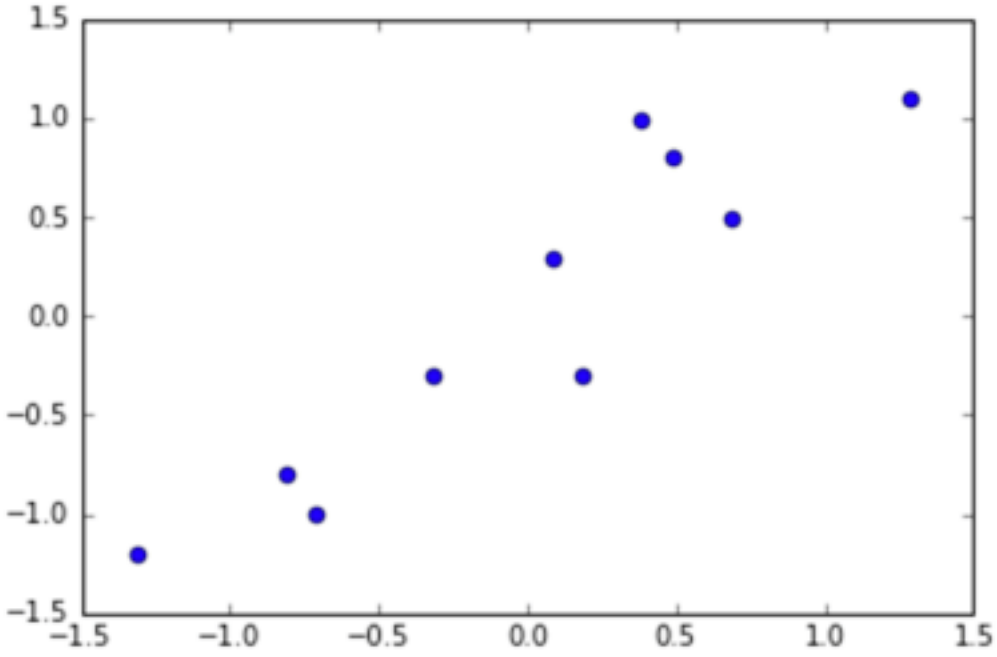
```
import numpy as np
x=np.array([2.5,0.5,2.2,1.9,3.1,2.3,2,1,1.5,1.1])
y=np.array([2.4,0.7,2.9,2.2,3,2.7,1.6,1.1,1.6,0.9])
```

归一化数据：

```
mean_x=np.mean(x)
mean_y=np.mean(y)
scaled_x=x-mean_x
scaled_y=y-mean_y
data=np.matrix([[scaled_x[i],scaled_y[i]] for i in range(len(scaled_x))])
```

绘制散点图查看数据分布：

```
import matplotlib.pyplot as plt
plt.plot(scaled_x,scaled_y,'o')
```



data distribution

求协方差矩阵：

```
cov=np.cov(scaled_x,scaled_y)
```

求协方差矩阵的特征值和特征向量：

```
eig_val, eig_vec = np.linalg.eig(cov)
```

求出特征向量后，我们需要选择降维后的数据维度 k（n 维数据降为 k 维数据），但我们的数据只有两维，所以只能降一维：

```
eig_pairs = [(np.abs(eig_val[i]), eig_vec[:,i]) for i in range(len(eig_val))]
eig_pairs.sort(reverse=True)
feature=eig_pairs[0][1]
```

转化得到降维后的数据：

```
new_data_reduced=np.transpose(np.dot(feature,np.transpose(data))
```

原文链接：https://deeplearning4j.org/eigenvector#a-beginners-guide-to-eigenvectors-pca-covariance-and-entropy