菊安酱的机器学习第12期

菊安酱的直播间：https://live.bilibili.com/14988341

每周一晚8:00 菊安酱和你不见不散哦~(^o^)/~

更新日期：2019-1-21

作者：菊安酱

课件内容说明：

- 本文为作者原创，转载请注明作者和出处
- 如果想获得此课件及录播视频，可扫描下方二维码，回复"k"进群
- 若有任何疑问，请给作者留言。

12期完整版课纲

直播时间：每周一晚8:00

直播内容：

时间	期数	算法
2018/11/05	第1期	k-近邻算法
2018/11/12	第2期	决策树
2018/11/19	第3期	朴素贝叶斯
2018/11/26	第4期	Logistic回归
2018/12/03	第5期	支持向量机
2018/12/10	第6期	AdaBoost 算法
2018/12/17	第7期	线性回归
2018/12/24	第8期	树回归
2018/12/28	第9期	K-均值聚类算法
2019/01/07	第10期	Apriori 算法
2019/01/14	第11期	FP-growth 算法
2019/01/21	第12期	降维算法之PCA&SVD

降维算法之PCA&SVD

一、为什么要进行降维?

大家都知道，在低维下，数据更容易处理，但是在通常情况下我们的数据并不是如此，往往会有很多的特征，进而就会出现很多问题：

(一) 多余的特征会影响或误导学习器

(二) 更多特征意味着更多参数需要调整，过拟合风险也越大

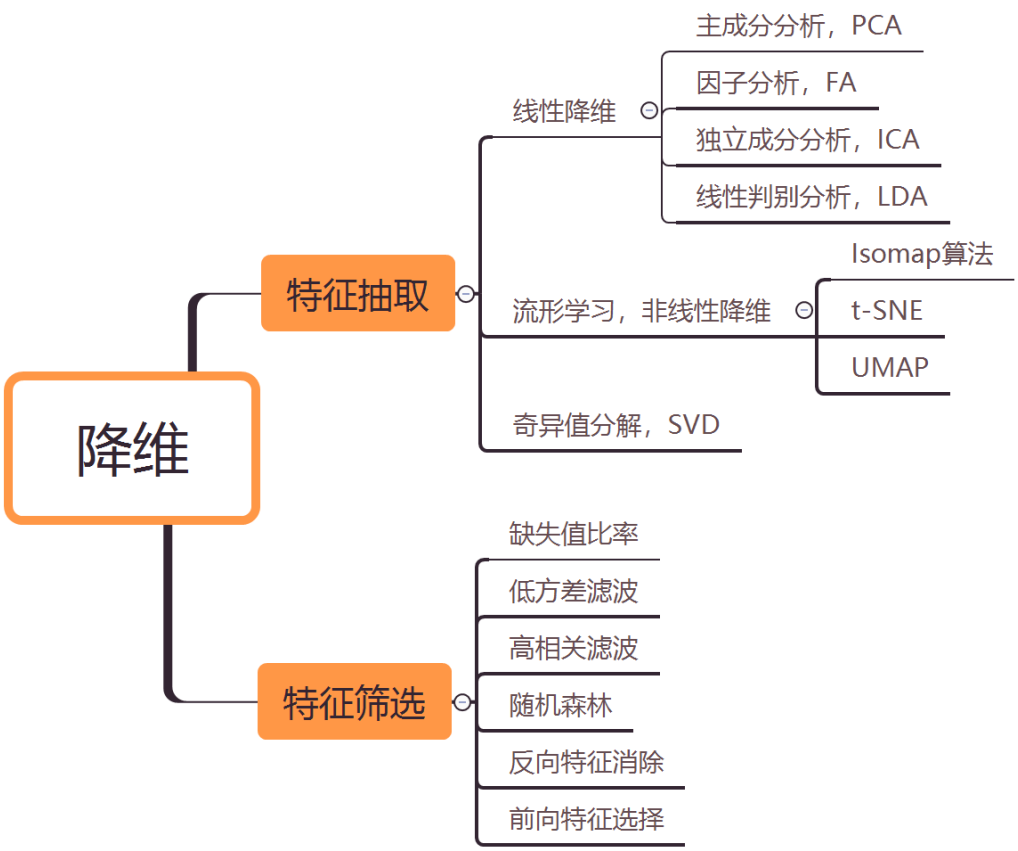
(三) 数据的维度可能只是虚高，真实维度可能比较小

(四) 维度越少意味着训练越快，更多东西可以尝试，能够得到更好的结果

(五) 如果我们想要可视化数据，就必须限制在两个或三个维度上

因此，我们需要通过降维（dimensionality reduction）把无关或冗余的特征删掉。

降维的方法主要有：



这一期的内容，我们主要讲解PCA和SVD。

二、主成分分析PCA

1. PCA概述

主成分分析（Principal Component Analysis）简称PCA。

PCA主要用于数据降维，对于一系列例子的特征组成的多维向量，多维向量里的某些元素本身没有区分性，比如某个元素在所有的例子中都为1，或者与1差距不大，那么这个元素本身就没有区分性，用它做特征来区分，贡献会非常小。所以我们的目的是找那些变化大的元素，即方差大的那些维，而去除掉那些变化不大的维，从而使特征留下的都是精品，而且计算量也变小了。

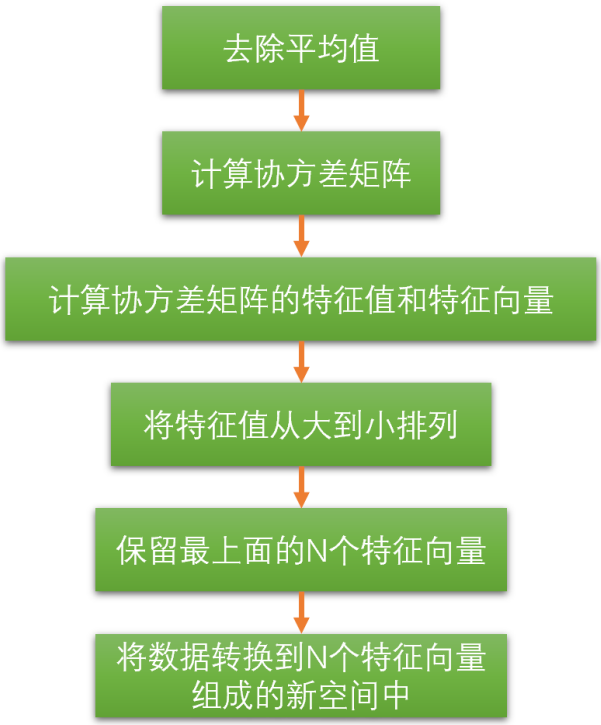
PCA降维流程：

在PCA中，数据从原来的坐标系转换到新的坐标系，由数据本身决定。转换坐标系时，以**方差最大的方向作为坐标轴方向**，因为数据的最大方差给出了数据的最重要的信息。第一个新坐标轴选择的是原始数据中方差最大的方法，第二个新坐标轴选择的是**与第一个新坐标轴正交且方差次大的方向**。重复该过程，**重复次数为原始数据的特征维数**。通过这种方式获得的新的坐标系，我们发现，大部分方差都包含在前面几个坐标轴中，后面的坐标轴所含的方差几乎为0。于是，我们可以忽略余下的坐标轴，只保留前面的几个含有绝大部分方差的坐标轴。事实上，这样也就相当于只保留包含绝大部分方差的维度特征，而忽略包含方差几乎为0的特征维度，也就实现了对数据特征的降维处理。

那么，我们如何得到这些包含最大差异性的主成分方向呢？事实上，通过计算数据矩阵的协方差矩阵，然后得到协方差矩阵的特征值及特征向量，选择特征值最大（也即包含方差最大）的N个特征所对应的特征向量组成的矩阵，我们就可以将数据矩阵转换到新的空间当中，实现数据特征的降维（N维）。

2. PCA的python实现

将数据转换成前N个主成分的过程为：



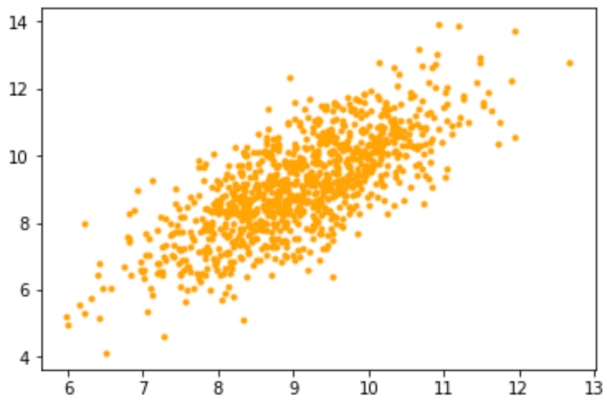
为什么要进行去除平均值呢？

我们进行PCA的主要目的是为了得到方差最大的前N个特征，为了减少计算量，我们第一步就将所有特征的均值变为0，来达到去除平均值的目的。

1. 导入原始数据集

```
#导入相应包
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

#导入数据
testSet = pd.read_table('testSet.txt',header=None)
testSet.head()
testSet.shape

#查看原始数据分布
plt.scatter(testSet.iloc[:,0],testSet.iloc[:,1],marker = '.',c='orange');
```

```
In [4]: plt.scatter(testSet.iloc[:,0],testSet.iloc[:,1],marker = '.',c='orange');
```



2. 编写PCA实现函数

首先，**去均值化**：

```
dataSet = testSet
#计算均值
meanVals = dataSet.mean(0)
meanVals
#去均值化，均值变为0
meanRemoved = dataSet - meanVals
meanRemoved
```

In [5]:
```
dataSet = testSet
meanVals = dataSet.mean(0)
meanVals
```

Out[5]:
```
0    9.063936
1    9.096002
dtype: float64
```

In [6]:
```
meanRemoved = dataSet - meanVals
meanRemoved.head()
```

Out[6]:

	0	1
0	1.171250	2.225995
1	1.058403	2.714991
2	0.126300	-0.191059
3	0.242435	0.751392
4	-0.733805	-0.755650

然后，**计算协方差**并将其变为矩阵，numpy中的cov()函数可以直接计算协方差

注意：cov(X,0) = cov(X) 除数是n-1(n为样本个数)，cov(X,1) 除数是n

```
#计算协方差矩阵
covMat = np.mat(np.cov(meanRemoved, rowvar=0))
covMat
```

In [145]:
```
covMat = np.mat(np.cov(meanRemoved, rowvar=0))
covMat
```

Out[145]:
```
matrix([[1.05198368, 1.1246314 ],
        [1.1246314 , 2.21166499]])
```

In [146]:
```
covMat.shape
```

Out[146]:
```
(2, 2)
```

计算方差矩阵的特征值和右特征向量

```
#计算方差矩阵的特征值和右特征向量
eigVals,eigVects = np.linalg.eig(covMat)
eigVals
eigVals.shape

eigVects
eigVects.shape
```

```
In [11]: eigVals,eigVects = np.linalg.eig(covMat)

In [12]: eigVals

Out[12]: array([0.36651371, 2.89713496])

In [13]: eigVals.shape

Out[13]: (2,)

In [14]: eigVects

Out[14]: matrix([[-0.85389096, -0.52045195],
                 [ 0.52045195, -0.85389096]])

In [15]: eigVects.shape

Out[15]: (2, 2)
```

对特征值进行排序，找出最大的N个特征值，这里的数据集是二维的，我们想把它降成一维，所以N=1

```
#对特征值排序，.argsort()函数默认从小到大排序
eigValInd = np.argsort(eigVals)
#提取出最大的N个特征
N=1
eigValInd = eigValInd[:-(N+1):-1]
eigValInd
```

根据特征值最大的N个特征值对应的索引，提取出相应的特征向量，组成**压缩矩阵**

```
redEigVects = eigVects[:,eigValInd]
redEigVects
```

```
In [14]: eigVects

Out[14]: matrix([[-0.85389096, -0.52045195],
                 [ 0.52045195, -0.85389096]])

In [15]: eigVects.shape

Out[15]: (2, 2)

In [16]: eigValInd = np.argsort(eigVals)
         N=1
         eigValInd = eigValInd[:-(N+1):-1]
         eigValInd

Out[16]: array([1], dtype=int64)

In [17]: redEigVects = eigVects[:,eigValInd]
         redEigVects

Out[17]: matrix([[-0.52045195],
                 [-0.85389096]])
```

将去除均值后的数据矩阵*压缩矩阵，**转换到新的空间**，使维度降低为N

```
lowDDataMat = np.mat(meanRemoved) * redEigVects #注意此处meanRemoved要转换成矩阵
lowDDataMat
```

In [19]:
```
lowDDataMat = np.mat(meanRemoved) * redEigVects
lowDDataMat[:10]
```

Out[19]:
```
matrix([[-2.51033597],
        [-2.86915379],
        [ 0.09741085],
        [-0.76778222],
        [ 1.02715333],
        [-1.44409178],
        [-2.17360352],
        [-0.7739988 ],
        [-1.09983463],
        [-1.70275987]])
```

利用降维后的矩阵反构出原数据矩阵(用作测试，可跟未压缩的原矩阵比对)

```
reconMat = (lowDDataMat * redEigVects.T) + np.mat(meanVals)
reconMat
reconMat.shape
```

In [20]:
```
reconMat = (lowDDataMat * redEigVects.T) + np.mat(meanVals)
reconMat
```

Out[20]:
```
matrix([[10.37044569, 11.23955536],
        [10.55719313, 11.54594665],
        [ 9.01323877,  9.01282393],
        ...,
        [ 9.32502753,  9.52436704],
        [ 9.0946364 ,  9.14637075],
        [ 9.16271152,  9.2580597 ]])
```

In [21]:
```
reconMat.shape
```

Out[21]:
```
(1000, 2)
```

3. 封装函数

将上述过程封装成一个函数，这个函数中有两个参数：一个是原始数据集，一个是可选参数（即应用的N个特征）如果不指定N的值，默认返回前999999个特征，或者原始数据集中的全部特征。

```
"""
函数功能：实现PCA降维
参数说明：
    dataSet：原始数据集
    N:希望将数据降为N维
```

```
    返回:
        lowDDataMat: 降维之后的矩阵
        reconMat: 重构的数据
    """
    def pca(dataSet, N=9999999):
        meanVals = dataSet.mean(0)
        meanRemoved = dataSet - meanVals
        covMat = np.mat(np.cov(meanRemoved, rowvar=0))
        eigVals,eigVects = np.linalg.eig(covMat)
        eigValInd = np.argsort(eigVals)
        eigValInd = eigValInd[:-(N+1):-1]
        redEigVects = eigVects[:,eigValInd]
        lowDDataMat = np.mat(meanRemoved) * redEigVects
        reconMat = (lowDDataMat * redEigVects.T) + np.mat(meanVals)
        return lowDDataMat, reconMat
```

运行函数，将二维数据降成一维数据：

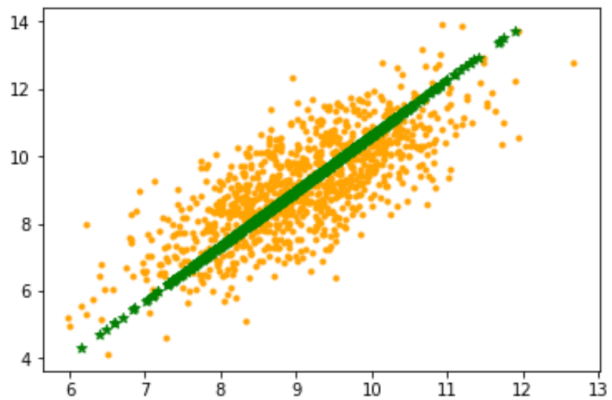
```
lowDDataMat, reconMat = pca(testSet, N=1)

#查看结果
lowDDataMat
lowDDataMat.shape

reconMat[:5]
testSet.head()
```

上述降维过程，首先根据数据矩阵的协方差的特征值和特征向量，得到最大的N个特征值对应的特征向量组成的矩阵，可以称之为压缩矩阵；得到了压缩矩阵之后，将去均值的数据矩阵乘以压缩矩阵，就实现了将原始数据特征转化为新的空间特征，进而使数据特征得到了压缩处理。

将原数据和压缩后的数据放在一张里面：

```
plt.scatter(testSet.iloc[:,0],testSet.iloc[:,1],marker = '.',c='orange')
plt.scatter(reconMat[:,0].A.flatten(),reconMat[:,1].A.flatten(), marker='*',c='g');
```

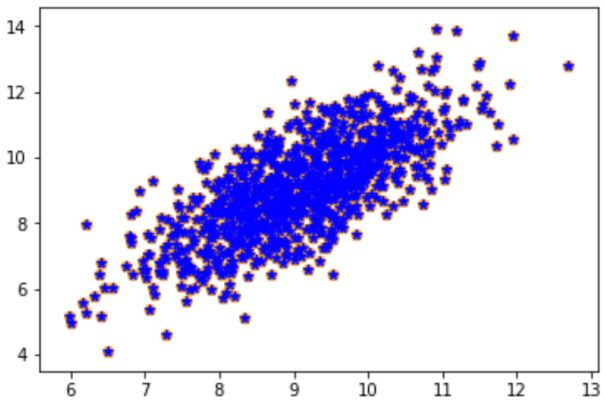
```
In [28]: plt.scatter(testSet.iloc[:,0],testSet.iloc[:,1],marker = '.',c='orange')
         plt.scatter(reconMat[:,0].A.flatten(),reconMat[:,1].A.flatten(), marker='*',c='g');
```



假设N=2，查看函数运行结果：

```
lowDDataMat, reconMat = pca(testSet, N=2)
lowDDataMat
reconMat

#将原数据和压缩后的数据放在一张里面：
plt.scatter(testSet.iloc[:,0],testSet.iloc[:,1],marker = 'o',c='orange')
plt.scatter(reconMat[:,0].A.flatten(),reconMat[:,1].A.flatten(), marker='*',c='b');
```

In [32]:
```
plt.scatter(testSet.iloc[:,0],testSet.iloc[:,1],marker = 'o',c='orange')
plt.scatter(reconMat[:,0].A.flatten(),reconMat[:,1].A.flatten(), marker='*',c='b');
```



3. 小案例：PCA对半导体数据进行降维

我们知道，像集成电路这样的半导体，成本非常昂贵。我们在面对大规模和高维度数据集时，显然计算损耗会很大，无疑会非常耗时。如果能在制造过程中尽早和尽快地检测出是否出现瑕疵，将可能为企业节省大量的成本和时间。所以，如果利用PCA等降维技术将高维的数据特征进行降维处理，保留那些最重要的数据特征，舍弃那些可以忽略的特征，将大大加快我们的数据处理速度和计算损耗，为企业节省不小的时间和成本。

导入数据集

半导体数据存储在secom.data文件中，由于该文件是以空格分隔，需要将sep=' '(中间为空格)。

```
secom = pd.read_table('secom.data',sep = ' ',header = None)
secom.head()
secom.shape
```

处理缺失值

大家从数据中可以看出，原始数据集中有很多缺失值，首先要处理这些缺失值，然后再进行降维处理。

```
#1.删除缺失值大于等于80%的特征
secom1=secom.copy()
nanInd = []
for i in range(secom1.shape[1]):
    nan = np.isnan(secom1.iloc[:,i]).mean()
    if nan*100>=80:
        print(f'第{i}列的缺失值比例为{round(nan*100,2)}%')
        nanInd.append(i)
```

```
secom1.drop(secom1.columns[nanInd],axis=1,inplace=True)

#查看处理结果
secom1.head()
secom1.shape

#2.缺失值比例小于80%的特征用均值填补缺失值
for i in secom1.columns:
    secom1[i].fillna(secom1[i].mean(),inplace = True)
```

数据探索

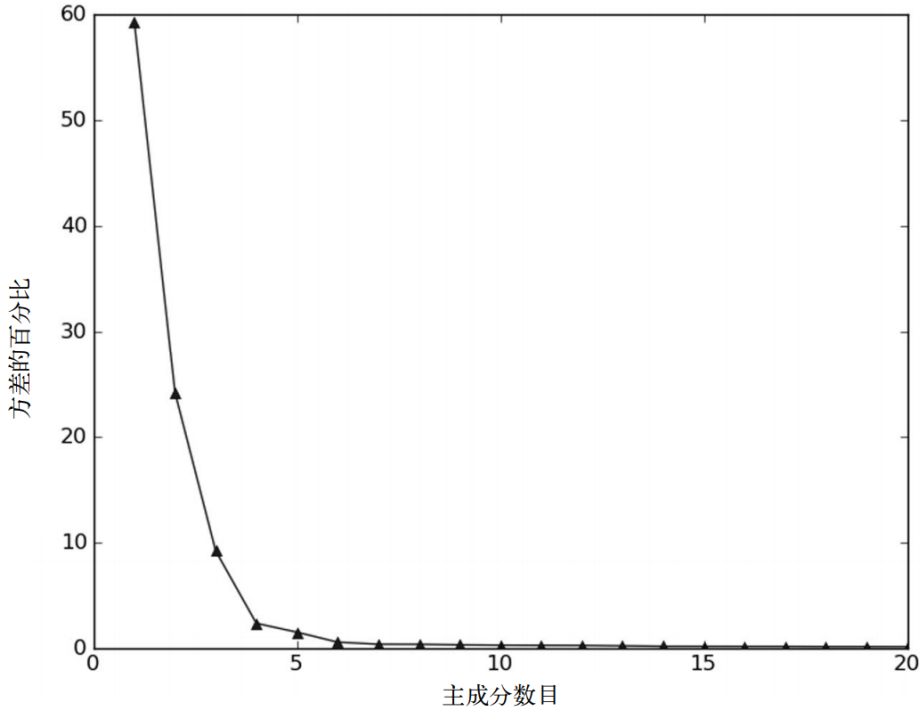
```
#计算均值
meanVals = secom1.mean()
meanVals

#去均值化
meanRemoved = secom1 - meanVals
meanRemoved

#计算协方差
covMat = np.mat(np.cov(meanRemoved, rowvar=0))
covMat
covMat.shape

#计算特征值和特征向量
eigVals,eigVects = np.linalg.eig(covMat)
eigVects.shape
eigVals.shape
eigVals
```

从特征值运行结果来看，大家肯定发现了：这里有很多特征值都为0（实际上将近20%的特征值均为0），这意味着这些特征都是其他特征的副本，也就是说，这些特征都可以通过其他特征来表示，而其本身并没有附带额外的信息。那些特征对于我们来说，就是无用的特征，可以直接将其舍去。

另外，大家可能也注意到了，这些特征值中还有一些很小的负值，这些负值出现的原因是数值误差应该四舍五入成0.

```
#计算特征值为0的个数
(eigVals==0).sum()

#计算特征值为0比例
(eigVals==0).sum()/len(eigVals)
```

下面，我们来看一下前20个主成分占总方差的百分比



可以看出，大部分方差都包含在前面的几个主成分中，舍弃后面的主成分并不会损失太多的信息。如果保留前6个主成分，则数据集可以从590个特征约简成6个特征，大概实现了100：1的压缩

再看一下这些主成分所对应的方差百分比和累积方差百分比。从累积方差百分比可以看出，前6个主成分就覆盖了总数据96.8%的方差，也就是说前6个主成分就包含了整个数据集96.8%的信息。

表13-1　半导体数据中前7个主成分所占的方差百分比

主　成　分	方差百分比（%）	累积方差百分比（%）
1	59.2	59.2
2	24.1	83.4
3	9.2	92.5
4	2.3	94.8
5	1.5	96.3
6	0.5	96.8
7	0.3	97.1
20	0.08	99.3

根据这些结果，大家可以知道数据集汇总大部分的信息都汇集在前面的多个主成分中。但是在实际应用中，具体要截取多少个主成分，或者说具体让数据集降到多少维，需要实际情况实际分析，有人可能只需要包含90%的信息就够了，也有人需要99%的信息。

三、奇异值分解SVD

1. SVD概述

奇异值分解（Singular Value Decomposition）简称SVD，主要作用是简化数据，提取信息。

利用SVD实现，我们能够用小得多的数据集来表示原始数据集。这样做，实际上是去除了噪声和冗余信息。当我们试图节省空间时，去除噪声和冗余信息就是很崇高的目标了，但是在这里我们则是从数据中抽取信息。基于这个视角，我们就可以把SVD看成是**从有噪声数据中抽取相关特征。**

相信大家都听说过二八原则。在很多情况下，二八原则对于数据携带信息同样适用，数据中可能仅仅只有20%的部分携带了数据集中的大部分信息（姑且有80%），那剩下80%的数据中可能包含的要么是噪声，要么是一些毫不相关的信息。

那SVD是如何从这些充满着大量噪声的数据中抽取相关特征呢?

我们先来看一下SVD的公式:

$$A_{m\times n} = U_{m\times m}\Sigma_{m\times n}V^T_{n\times n}$$

这个公式中， U 和 V 都是正交矩阵，即:

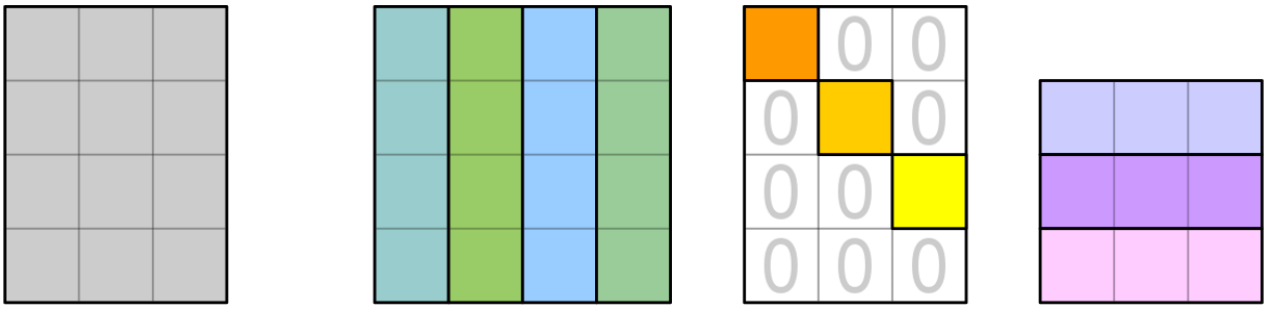
$$U^TU = E_{m\times m}$$
$$V^TV = E_{n\times n}$$

原始数据集A是一个m行n列的矩阵，它被分解成了三个矩阵，分别是:

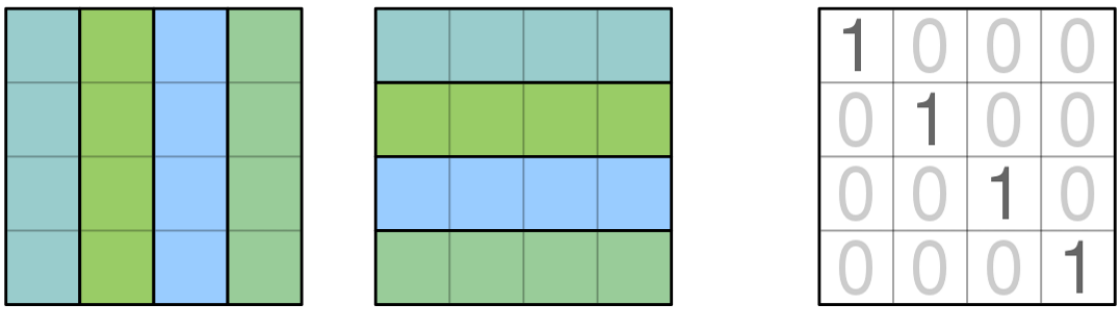
矩阵	别称	维度	计算方式
U矩阵	A的左奇异向量	m行m列	列由AA^T 的特征向量构成，且特征向量为**单位列向量**
Σ矩阵	A的奇异值	m行n列	对角元素来源于 AA^T或 A^TA 的特征值的平方根， 并且是按**从大到小的顺序**排列的。值越大可以理解为越重要。
V矩阵	A的右奇异向量	n行n列	列由 ATA 的特征向量构成，且特征向量为**单位列向量**

这个公式用到的就是**矩阵分解**技术。在线性代数中还有很多矩阵分解技术。矩阵分解可以将原始矩阵表示成新的易于处理的形式，这种新形式是两个或多个矩阵的乘积。我们可以将这种分解过程想象成代数中的因子分解。如何将12分解成两个数的乘积？(1,12)、(2,6)和(3,4)都是合理的答案。

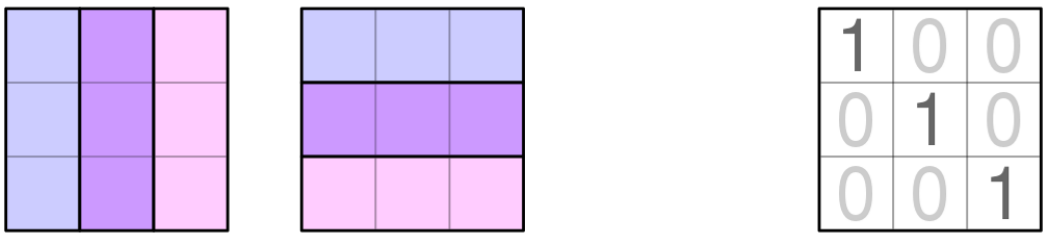
不同的矩阵分解技术具有不同的性质，其中有些更适合于某个应用，有些则更适合于其他应用。最常见的一种矩阵分解技术就是SVD。



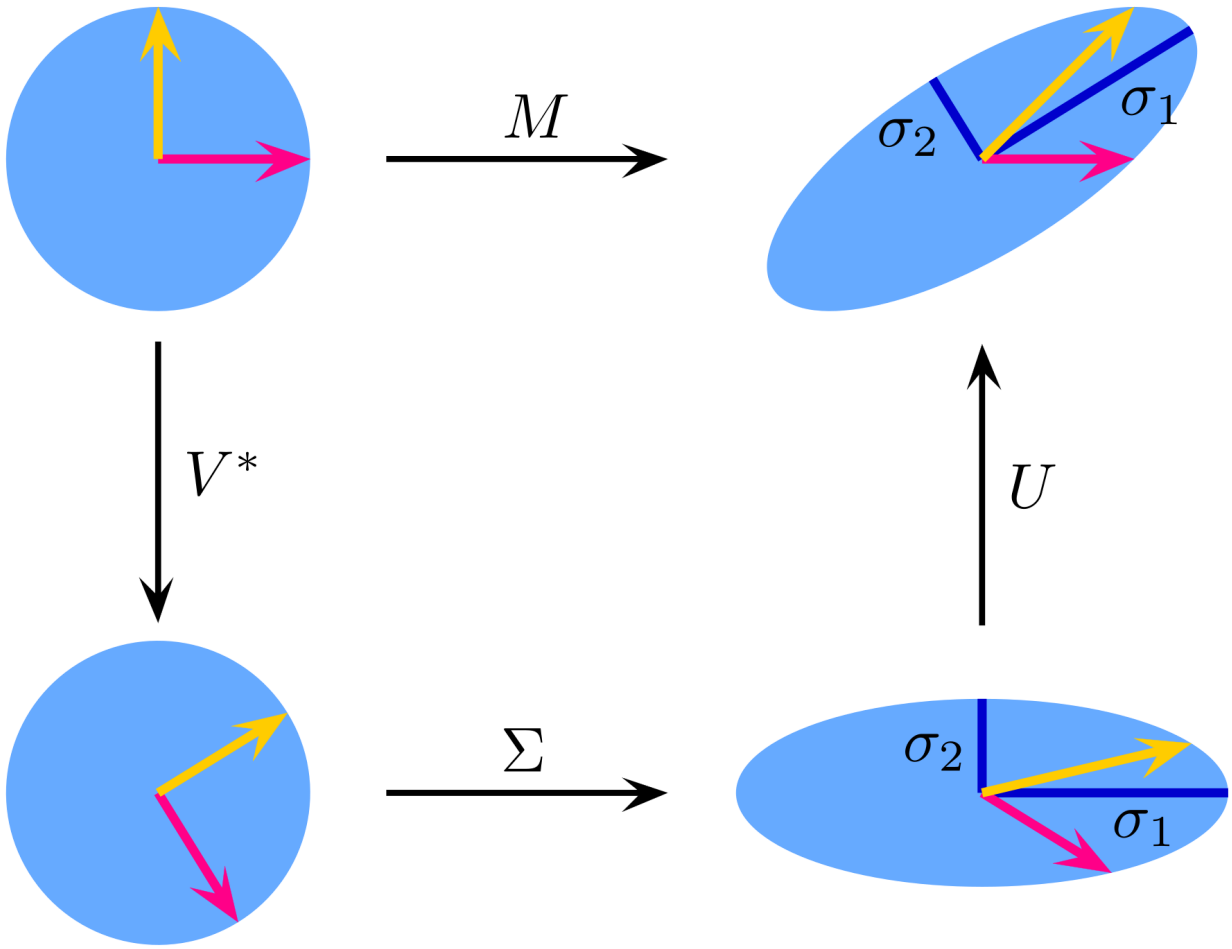
$$\underset{m\times n}{\mathbf{M}} = \underset{m\times m}{\mathbf{U}} \quad \underset{m\times n}{\mathbf{\Sigma}} \quad \underset{n\times n}{\mathbf{V}^*}$$



$$\mathbf{U}\quad \mathbf{U}^* = \mathbf{I}_m$$



$$\mathbf{V}\quad \mathbf{V}^* = \mathbf{I}_n$$



$$M = U \cdot \Sigma \cdot V^*$$

2. SVD的应用

2.1 信息检索

最早的SVD应用之一就是信息检索。利用SVD方法为隐形语义索引（Latent Semantic Indexing，LSI）或者隐形语义分析（Latent Semantic Analysis，LSA）。

在LSI中，一个矩阵是由文档和词语组成的。当我们在该矩阵上应用SVD时，就会构建出多个奇异值。这些奇异值代表了文档中的概念或主题，这一特点可以用于更高效的文档搜索。在词语拼写错误时，只基于词语存在与否的简单搜索方法会遇到问题。简单搜索的另一个问题就是同义词的使用。这就是说，当我们查找一个词时，其同义词所在的文档可能并不会匹配上。如果我们从上千篇相似的文档中抽取出概念，那么同义词就会映射为同一概念。 这样就可以大大提高文档搜索的效率。

2.2 推荐系统

SVD的另外一个应用就是推荐系统。也是目前SVD最主要的一个应用简单版本的推荐系统能够计算项或者人之间的相似度。更先进的方法则先利用SVD从数据中构建一个主题空间，然后再在该空间下计算其相似度。

小故事

Netflix(Nasdaq NFLX) 成立于1997年，是一家**在线影片租赁提供商**，主要提供Netflix超大数量的DVD并免费递送，总部位于美国加利福尼亚州的洛斯盖图。
NETFLIX 设计开发了一个**电影推荐系统**叫 Cinematch，这是一个智能预测系统，它能够根据用户以前的评分数据预测到这位顾客可能喜欢什么样的主题和风格，等新电影发行后马上为相应的用户群 体进行推荐。可是这个推荐系统的智能水平有限，准确度不高，不能令人满意。
2006年， NETFLIX 对外宣布，他们要设立一项大赛，公开征集电影推荐系统的较佳电脑算法，第一个能把现有推荐系统的**准确率提高 10%** 的参赛者将获得**一百万美元**的奖金。
2009 年 9 月 21 日，来自全世界 186 个国家的四万多个参赛团队经过近三年的较量，终于有了结果。一个由工程师和统计学家组成的七人团队夺得了大奖。而最后的获奖者就使用了**SVD**来做的推荐系统。

【完整版】3. 基于协同过滤的推荐系统

【完整版】3.1 相似度计算

【完整版】3.2 基于物品的相似度

【完整版】3.3 基于用户的相似度

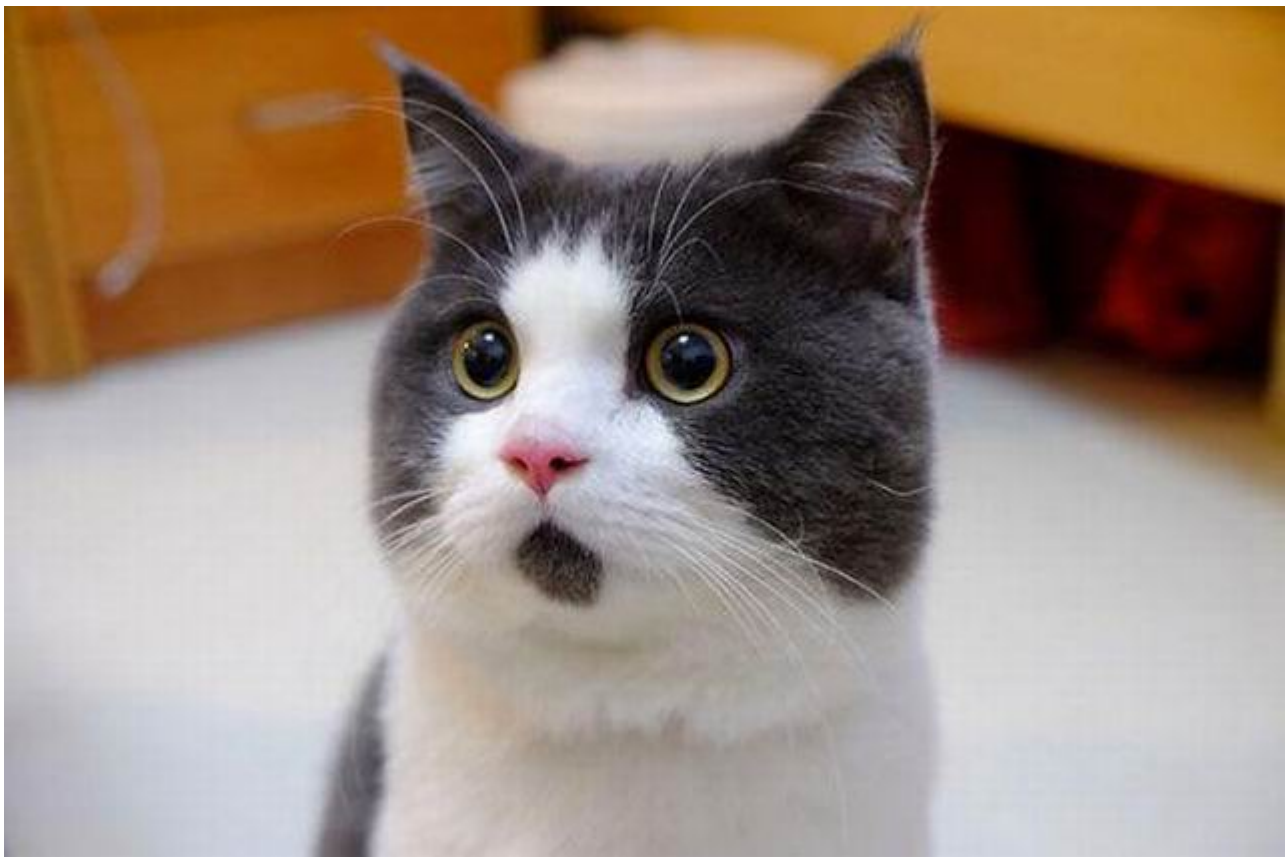
【完整版】4. 案例：餐馆菜肴推荐系统

【完整版】4.1 给用户推荐未品尝过的菜肴

【完整版】4.2 利用SVD提高推荐的效果

【完整版】4.3 构建推荐系统面临的挑战

【完整版】5. 案例：基于SVD的图像压缩



其他

- 菊安酱的直播间：https://live.bilibili.com/14988341
- 菊安酱所有的直播结束啦~，谢谢大家三个月来的陪伴~