

第9章 降维

内容提要

□ 主成分分析 (PCA)

□ 对半导体数据进行降维处理

9.1 降维技术

降维就是一种对高维度特征数据预处理方法。降维是将高维度的数据保留下最重要的一些特征，去除噪声和不重要的特征，从而实现提升数据处理速度的目的。在实际的生产和应用中，降维在一定的信息损失范围内，可以为我们节省大量的时间和成本。降维也成为应用非常广泛的数据预处理方法。

降维具有如下一些优点：

- 使得数据集更易使用。
- 降低算法的计算开销。
- 去除噪声。
- 使得结果容易理解。

在已标注与未标注的数据上都有降维技术。这里我们将主要关注未标注数据上的降维技术，该技术同时也可以应用于已标注的数据。

降维的算法有很多，比如奇异值分解 (SVD)、主成分分析 (PCA)、因子分析 (FA)、独立成分分析 (ICA)、线性判别分析 (LDA)。

PCA(Principal Component Analysis)，即主成分分析方法，是一种使用最广泛的数据降维算法。PCA 的主要思想是将 n 维特征映射到 k 维上，这 k 维是全新的正交特征也被称为主成分，是在原有 n 维特征的基础上重新构造出来的 k 维特征。PCA 的工作就是从原始的空间中顺序地找一组相互正交的坐标轴，新的坐标轴的选择与数据本身是密切相关的。其中，第一个新坐标轴选择是原始数据中方差最大的方向，第二个新坐标轴选取是与第一个坐标轴正交的平面中使得方差最大的，第三个轴是与第 1,2 个轴正交的平面中方差最大的。依次类推，可以得到 n 个这样的坐标轴。通过这种方式获得的新的坐标轴，我们发现，大部分方差都包含在前面 k 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 k 个含有绝大部分方差的坐标轴。事实上，这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，实现对数据特征的降维处理。

9.1.1 移动坐标轴

考虑一下图9.1中的大量数据点。如果要求我们画出一条直线，这条线要尽可能覆盖这些点，那么最长的线可能是哪条？我做过多次尝试。在图9.1中，3 条直线中 B 最长。在 PCA 中，我们对数据的坐标进行了旋转，该旋转的过程取决于数据的本身。第一条坐标轴旋转到覆盖数据的最大

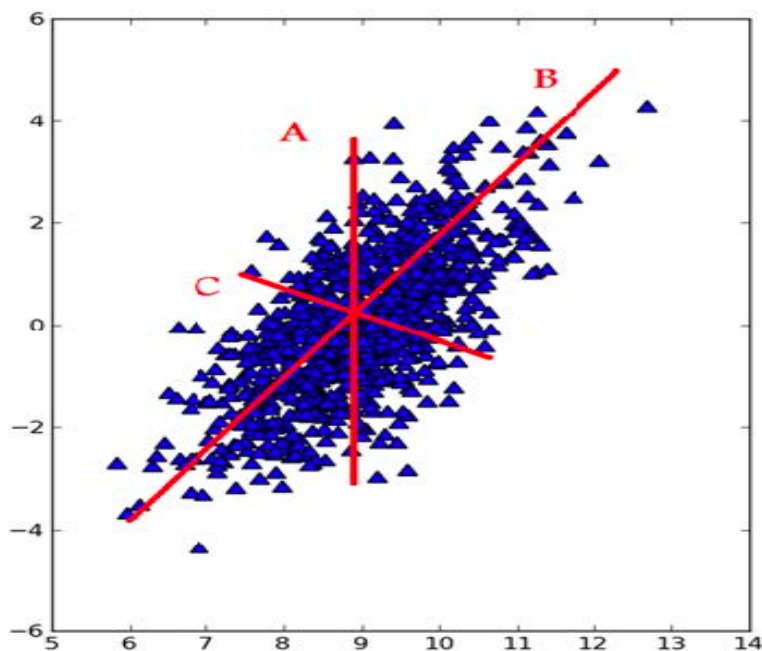



图 9.1: 覆盖整个数据集的三条直线，其中直线 B 最长，并给出了数据集中差异化最大的方向

我们已经实现了坐标轴的旋转，接下来开始讨论降维。坐标轴的旋转并没有减少数据的维度。考虑图9.2，其中包含着 3 个不同的类别。要区分这 3 个类别，可以使用决策树。我们还记得决策树每次都是基于一个特征来做决策的。我们会发现，在 x 轴上可以找到一些值，这些值能够很好地将这 3 个类别分开。这样，我们就可能得到一些规则，比如当 $(X < 4)$ 时，数据属于类别 0。如果使用 SVM 这样稍微复杂一点的分分类器，我们就会得到更好的分类面和分类规则，比如当 $(w_0 \times x + w_1 \times y + b) > 0$ 时，数据也属于类别 0。SVM 可能比决策树得到更好的分类间隔，但是分类超平面却很难解释。

通过 PCA 进行降维处理，我们就可以同时获得 SVM 和决策树的优点：一方面，得到了和决策树一样简单的分类器，同时分类间隔和 SVM 一样好。考察图9.2中下面的图，其中的数据来自于上面的图并经 PCA 转换之后绘制而成的。如果仅使用原始数据，那么这里的间隔会比决策树的间隔更大。另外，由于只需要考虑一维信息，因此数据就可以通过比 SVM 简单得多的很容易采用的规则进行区分。

在图9.2中，我们只需要一维信息即可，因为另一维信息只是对分类缺乏贡献的噪声数据。在二维平面下，这一点看上去微不足道，但是如果在高维空间下则意义重大。我们已经对 PCA 的基本过程做出了简单的阐述，接下来就可以通过代码来实现 PCA 过程。前面我曾提到的第一个主成分就是从数据差异性最大（即方差最大）的方向提取出来的，第二个主成分则来自于数据差异性次大的方向，并且该方向与第一个主成分方向正交。通过数据集的协方差矩阵及其特征值分析，我们就可以求得这些主成分的值。一旦得到了协方差矩阵的特征向量，我们就可以保留最大的 N 个值。这些特征向量也给出了 N 个最重要特征的真实结构。我们可以通过将数据乘上这 N 个特征向量而将它转换到新的空间。

 **笔记** 特征值分析是线性代数中的一个领域，它能够通过数据的一般格式来揭示数据的“真实”结构，即我们常说的特征向量和特征值。在等式 $Av = \lambda v$ 中， v 是特征向量， λ 是特征值。特征值都是简单的标量值，因此 $Av = \lambda v$ 代表的是：如果特征向量 v 被某个矩阵 A 左乘，那么

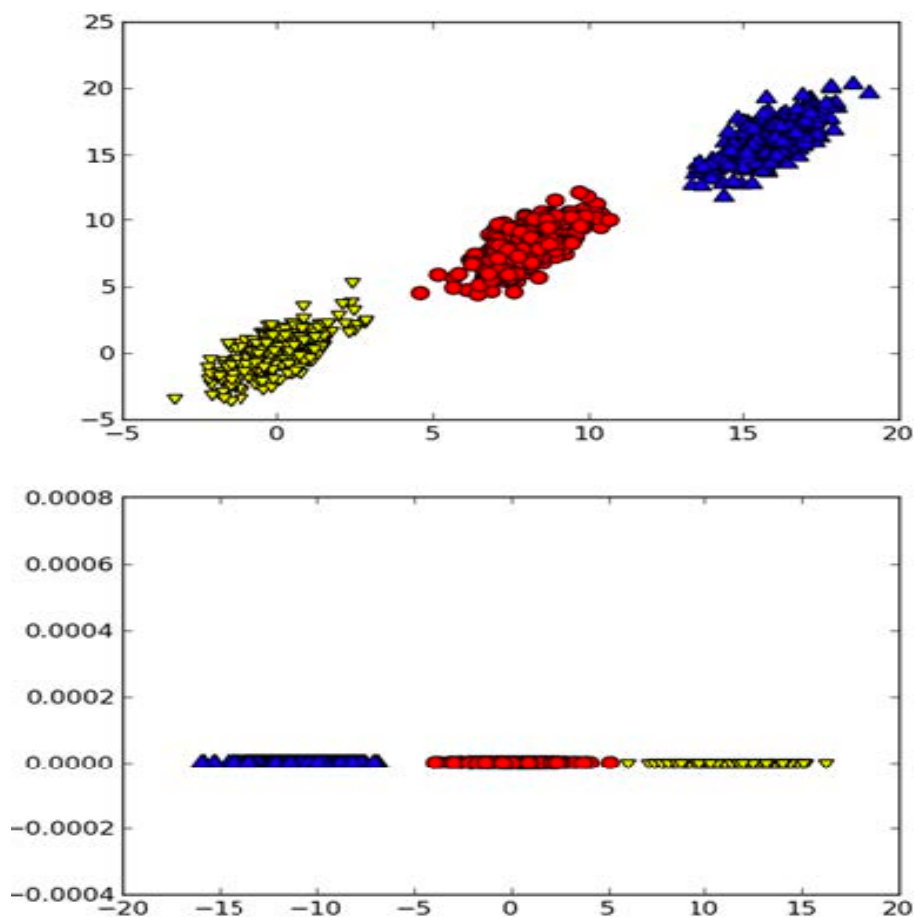


图 9.2: 二维空间的 3 个类别。当在该数据集上应用 PCA 时, 就可以去掉一维, 从而使得该分类问题变得更容易处理

它就等于某个标量 λ 乘以 v 。幸运的是, NumPy 中有寻找特征向量和特征值的模块 `linalg`, 它有 `eig()` 方法, 该方法用于求解特征向量和特征值。

9.1.2 在 NumPy 中实现 PCA

将数据转换成前 N 个主成分的伪码大致如下:

```
去除平均值
计算协方差矩阵
计算协方差矩阵的特征值和特征向量
将特征值从大到小排序
保留最上面的N个特征向量
将数据转换到上述N个特征向量构建的新空间中
```

```
from numpy import *

def loadDataSet(fileName, delim='\t'):
    fr = open(fileName)
    stringArr = [line.strip().split(delim) for line in fr.readlines()]
    datArr = [map(float,line) for line in stringArr]
    return mat(datArr)
```

```
def pca(dataMat, topNfeat=99999999):
    meanVals = mean(dataMat, axis=0)
    meanRemoved = dataMat - meanVals #remove mean
    covMat = cov(meanRemoved, rowvar=0)
    eigVals,eigVects = linalg.eig(mat(covMat))
    eigValInd = argsort(eigVals)          #sort, sort goes smallest to largest
    eigValInd = eigValInd[:-(topNfeat+1):-1] #cut off unwanted dimensions
    redEigVects = eigVects[:,eigValInd]   #reorganize eig vects largest to smallest
    lowDDataMat = meanRemoved * redEigVects#transform data into new dimensions
    reconMat = (lowDDataMat * redEigVects.T) + meanVals
    return lowDDataMat, reconMat
```

pca() 函数有两个参数：第一个参数是用于进行 PCA 操作的数据集，第二个参数 topNfeat 则是一个可选参数，即应用的 N 个特征。如果不指定 topNfeat 的值，那么函数就会返回前 9 999 999 个特征，或者原始数据中全部的特征。首先计算并减去原始数据集的平均值。然后，计算协方差矩阵及其特征值，接着利用 argsort() 函数对特征值进行从小到大的排序。根据特征值排序结果的逆序就可以得到 topNfeat 个最大的特征向量。这些特征向量将构成后面对数据进行转换的矩阵，该矩阵则利用 N 个特征将原始数据转换到新空间中。最后，原始数据被重构后返回用于调试，同时降维之后的数据集也被返回了。

```
import matplotlib.pyplot as plt
import matplotlib

fig = plt.figure()
ax = fig.add_subplot(111)

ax.scatter(dataMat[:,0].flatten().A[0], dataMat[:,1].flatten().A[0],
marker='^', s=90)

ax.scatter(reconMat[:,0].flatten().A[0], reconMat[:,1].flatten().A[0],
marker='o', s=50, c='red')
```

9.2 示例：利用 PCA 对半导体制造数据降维

半导体是在一些极为先进的工厂中制造出来的。工厂或制造设备不仅需要花费上亿美元，而且还需要大量的工人。制造设备仅能在几年内保持其先进性，随后就必须更换了。单个集成电路的加工时间会超过一个月。在设备生命期有限，花费又极其巨大的情况下，制造过程中的每一秒钟都价值巨大。如果制造过程中存在瑕疵，我们就必须尽早发现，从而确保宝贵的时间不会花费在缺陷产品的生产上。

一些工程上的通用解决方案是通过早期测试和频繁测试来发现有缺陷的产品，但仍然有一些存在瑕疵的产品通过了测试。如果机器学习技术能够用于进一步减少错误，那么它就会为制造商节省大量的资金。

接下来我们将考察面向上述任务中的数据集，该数据集从[学在浙大](#)上下载。

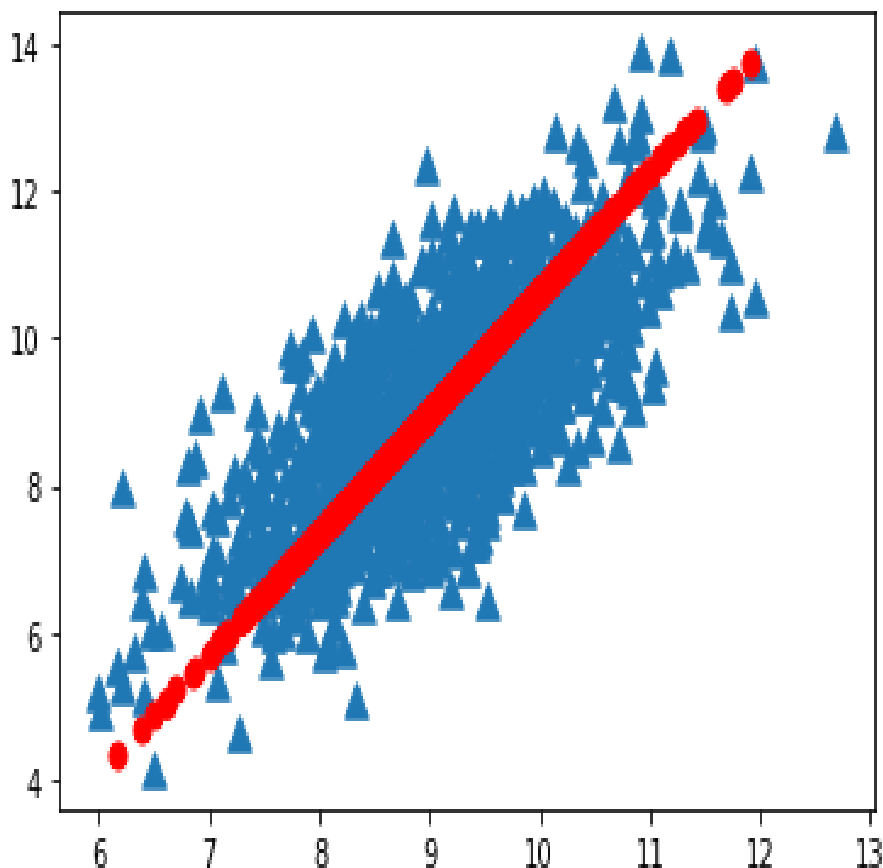


图 9.3: 原始数据集（三角形点表示）及第一主成分（圆形点表示）

该数据集有 590 个特征，包含很多缺失值。这些缺失值是以 NaN（Not a Number 的缩写）标识的。

在 590 个特征下，几乎所有样本都有 NaN，因此去除不完整的样本不太现实。尽管我们可以将所有的 NaN 替换成 0，但是由于并不知道这些值的意义，所以这样做是个下策。如果它们是开氏温度，那么将它们置成 0 这种处理策略就太差劲了。下面我们用平均值来代替缺失值，平均值根据那些非 NaN 得到。

```
def replaceNaNWithMean():
    datMat = loadDataSet('secom.data', ' ')
    numFeat = shape(datMat)[1]
    for i in range(numFeat):

        dataMat_i = datMat[:,i].A
        dataMat_i = dataMat_i.astype(np.float)
        meanVal = mean(datMat[nonzero(~isnan(dataMat_i))[0],i]) #values that are not NaN (a number)
        datMat[nonzero(isnan(dataMat_i))[0],i] = meanVal #set NaN values to mean

    return datMat
```

上述代码首先打开了数据集并计算出了其特征的数目，然后再在所有的特征上进行循环。对于每个特征，首先计算出那些非 NaN 值的平均值。然后，将所有 NaN 替换为该平均值。

我们已经去除了所有 NaN，接下来考虑在该数据集上应用 PCA。首先确认所需特征和可

以去除特征的数目。PCA 会给出数据中所包含的信息量。需要特别强调的是，数据（data）和信息（information）之间具有巨大的差别。数据指的是接受的原始材料，其中可能包含噪声和不相关信息。信息是指数据中的相关部分。这些并非只是抽象概念，我们还可以定量地计算数据中所包含的信息并决定保留的比例。

下面看看该如何实现这一点。首先，将数据集中所有的 NaN 替换成平均值：

```
dataMat = replaceNaNWithMean()
```

接下来从 `pca()` 函数中借用一些代码来达到我们的目的，之所以借用是因为我们想了解中间结果而非最后输出结果。首先调用如下语句去除均值：

```
meanVals = mean(dataMat, axis=0)
meanRemoved = dataMat - meanVals
```

然后计算协方差矩阵：

```
covMat = cov(meanRemoved, rowvar=0)
```

最后对该矩阵进行特征值分析：

```
eigVals,eigVects = linalg.eig(mat(covMat))
```

现在，我们可以观察一下特征值的结果：

```
array([ 5.34151979e+07, 2.17466719e+07, 8.24837662e+06, 2.07388086e+06,
        1.31540439e+06, 4.67693557e+05, 2.90863555e+05, 2.83668601e+05,
        2.37155830e+05, 2.08513836e+05, 1.96098849e+05, 1.86856549e+05,
        1.52422354e+05, 1.13215032e+05, 1.08493848e+05, 1.02849533e+05,
        1.00166164e+05, 8.33473762e+04, 8.15850591e+04, 7.76560524e+04,
        6.66060410e+04, 6.52620058e+04, 5.96776503e+04, 5.16269933e+04,
        5.03324580e+04, 4.54661746e+04, 4.41914029e+04, 4.15532551e+04,
        3.55294040e+04, 3.31436743e+04, 2.67385181e+04, 1.47123429e+04,
        1.44089194e+04, 1.09321187e+04, 1.04841308e+04, 9.48876548e+03,
        8.34665462e+03, 7.22765535e+03, 5.34196392e+03, 4.95614671e+03,
        4.23060022e+03, 4.10673182e+03, 3.41199406e+03, 3.24193522e+03,
        2.74523635e+03, 2.35027999e+03, 2.16835314e+03, 1.86414157e+03,
        1.76741826e+03, 1.70492093e+03, 1.66199683e+03, 1.53948465e+03,
        1.33096008e+03, 1.25591691e+03, 1.15509389e+03, 1.12410108e+03,
        1.03213798e+03, 1.00972093e+03, 9.50542179e+02, 9.09791361e+02,
        8.32001551e+02, 8.08898242e+02, 7.37343627e+02, 6.87596830e+02,
        5.64452104e+02, 5.51812250e+02, 5.37209115e+02, 4.93029995e+02,
        4.13720573e+02, 3.90222119e+02, 3.37288784e+02, 3.27558605e+02,
        3.08869553e+02, 2.46285839e+02, 2.28893093e+02, 1.96447852e+02,
        1.75559820e+02, 1.65795169e+02, 1.56428052e+02, 1.39671194e+02,
        1.28662864e+02, 1.15624070e+02, 1.10318239e+02, 1.08663541e+02,
        1.00695416e+02, 9.80687852e+01, 8.34968275e+01, 7.53025397e+01,
        6.89260158e+01, 6.67786503e+01, 6.09412873e+01, 5.30974002e+01,
        4.71797825e+01, 4.50701108e+01, 4.41349593e+01, 4.03313416e+01,
        3.95741636e+01, 3.74000035e+01, 3.44211326e+01, 3.30031584e+01,
        3.03317756e+01, 2.88994580e+01, 2.76478754e+01, 2.57708695e+01,
        2.44506430e+01, 2.31640106e+01, 2.26956957e+01, 2.16925102e+01,
```


2.10114869e+01, 2.00984697e+01, 1.86489543e+01, 1.83733216e+01,
 1.72517802e+01, 1.60481189e+01, 1.54406997e+01, 1.48356499e+01,
 1.44273357e+01, 1.42318192e+01, 1.35592064e+01, 1.30696836e+01,
 1.28193512e+01, 1.22093626e+01, 1.15228376e+01, 1.12141738e+01,
 1.02585936e+01, 9.86906139e+00, 9.58794460e+00, 9.41686288e+00,
 9.20276340e+00, 8.63791398e+00, 8.20622561e+00, 8.01020114e+00,
 7.53391290e+00, 7.33168361e+00, 7.09960245e+00, 7.02149364e+00,
 6.76557324e+00, 6.34504733e+00, 6.01919292e+00, 5.81680918e+00,
 5.44653788e+00, 5.12338463e+00, 4.79593185e+00, 4.47851795e+00,
 4.50369987e+00, 4.27479386e+00, 3.89124198e+00, 3.56466892e+00,
 3.32248982e+00, 2.97665360e+00, 2.61425544e+00, 2.31802829e+00,
 2.17171124e+00, 1.99239284e+00, 1.96616566e+00, 1.88149281e+00,
 1.79228288e+00, 1.71378363e+00, 1.68028783e+00, 1.60686268e+00,
 1.47158244e+00, 1.40656712e+00, 1.37808906e+00, 1.27967672e+00,
 1.22803716e+00, 1.18531109e+00, 9.38857180e-01, 9.18222054e-01,
 8.26265393e-01, 7.96585842e-01, 7.74597255e-01, 7.14002770e-01,
 6.79457797e-01, 6.37928310e-01, 6.24646758e-01, 5.34605353e-01,
 4.60658687e-01, 4.24265893e-01, 4.08634622e-01, 3.70321764e-01,
 3.67016386e-01, 3.35858033e-01, 3.29780397e-01, 2.94348753e-01,
 2.84154176e-01, 2.72703994e-01, 2.63265991e-01, 2.45227786e-01,
 2.25805135e-01, 2.22331919e-01, 2.13514673e-01, 1.93961935e-01,
 1.91647269e-01, 1.83668491e-01, 1.82518017e-01, 1.65310922e-01,
 1.57447909e-01, 1.51263974e-01, 1.39427297e-01, 1.32638882e-01,
 1.28000027e-01, 1.13559952e-01, 1.12576237e-01, 1.08809771e-01,
 1.07136355e-01, 8.60839655e-02, 8.50467792e-02, 8.29254355e-02,
 7.03701660e-02, 6.44475619e-02, 6.09866327e-02, 6.05709478e-02,
 5.93963958e-02, 5.22163549e-02, 4.92729703e-02, 4.80022983e-02,
 4.51487439e-02, 4.30180504e-02, 4.13368324e-02, 4.03281604e-02,
 3.91576587e-02, 3.54198873e-02, 3.31199510e-02, 3.13547234e-02,
 3.07226509e-02, 2.98354196e-02, 2.81949091e-02, 2.49158051e-02,
 2.36374781e-02, 2.28360210e-02, 2.19602047e-02, 2.00166957e-02,
 1.86597535e-02, 1.80415918e-02, 1.72261012e-02, 1.60703860e-02,
 1.49566735e-02, 1.40165444e-02, 1.31296856e-02, 1.21358005e-02,
 1.07166503e-02, 1.01045695e-02, 9.76055340e-03, 9.16740926e-03,
 8.78108857e-03, 8.67465278e-03, 8.30918514e-03, 8.05104488e-03,
 7.56152126e-03, 7.31508852e-03, 7.26347037e-03, 6.65728354e-03,
 6.50769617e-03, 6.28009879e-03, 6.19160730e-03, 5.64130272e-03,
 5.30195373e-03, 5.07453702e-03, 4.47372286e-03, 4.32543895e-03,
 4.22006582e-03, 3.97065729e-03, 3.75292740e-03, 3.64861290e-03,
 3.38915810e-03, 3.27965962e-03, 3.06633825e-03, 2.99206786e-03,
 2.83586784e-03, 2.74987243e-03, 2.31066313e-03, 2.26782347e-03,
 1.82206662e-03, 1.74955624e-03, 1.69305161e-03, 1.66624597e-03,
 1.55346749e-03, 1.51278404e-03, 1.47296800e-03, 1.33617458e-03,
 1.30517592e-03, 1.24056353e-03, 1.19823961e-03, 1.14381059e-03,
 1.13027458e-03, 1.11081803e-03, 1.08359152e-03, 1.03517496e-03,
 1.00164593e-03, 9.50024604e-04, 8.94981182e-04, 8.74363843e-04,
 7.98497544e-04, 7.51612221e-04, 6.63964303e-04, 6.21097650e-04,
 6.18098606e-04, 5.72611405e-04, 5.57509231e-04, 5.47002382e-04,
 5.27195077e-04, 5.11487997e-04, 4.87787872e-04, 4.74249071e-04,

4.52367688e-04, 4.24431101e-04, 4.19119025e-04, 3.72489906e-04,
 3.38125455e-04, 3.34002143e-04, 2.97951371e-04, 2.84845901e-04,
 2.79038288e-04, 2.77054476e-04, 2.54815125e-04, 2.67962796e-04,
 2.29230595e-04, 1.99245436e-04, 1.90381389e-04, 1.84497913e-04,
 1.77415682e-04, 1.68160613e-04, 1.63992030e-04, 1.58025553e-04,
 1.54226003e-04, 1.46890640e-04, 1.46097434e-04, 1.40079892e-04,
 1.35736724e-04, 1.22704035e-04, 1.16752515e-04, 1.14080847e-04,
 1.04252870e-04, 9.90265095e-05, 9.66039063e-05, 9.60766570e-05,
 9.16166344e-05, 9.07003475e-05, 8.60212633e-05, 8.32654025e-05,
 7.70526076e-05, 7.36470021e-05, 7.24998308e-05, 6.80209909e-05,
 6.68682699e-05, 6.14500429e-05, 5.99843179e-05, 5.49918002e-05,
 5.24646953e-05, 5.13403848e-05, 5.02336261e-05, 4.89288514e-05,
 4.51104474e-05, 4.29823766e-05, 4.18869715e-05, 4.14341559e-05,
 3.94822845e-05, 3.80307292e-05, 3.57776535e-05, 3.43901591e-05,
 2.98089203e-05, 2.72388358e-05, 1.91217363e-05, 1.96208174e-05,
 2.42608885e-05, 2.14440814e-05, 2.30962279e-05, 2.27807559e-05,
 1.88276186e-05, 1.66549051e-05, 1.46846459e-05, 1.43753346e-05,
 1.39779892e-05, 1.21760519e-05, 1.20295835e-05, 8.34247990e-06,
 1.13426750e-05, 1.09258905e-05, 8.93991858e-06, 9.23630205e-06,
 1.02782991e-05, 1.01021808e-05, 9.64538297e-06, 9.72678795e-06,
 7.36188590e-06, 7.20354827e-06, 6.69282813e-06, 6.49477814e-06,
 5.91044557e-06, 6.00244890e-06, 5.67034892e-06, 5.31392220e-06,
 5.09342484e-06, 4.65422046e-06, 4.45482134e-06, 4.11265577e-06,
 3.48065952e-06, 3.65202837e-06, 3.77558986e-06, 2.78847699e-06,
 2.57492503e-06, 2.66299627e-06, 2.39210232e-06, 2.06298821e-06,
 2.00824521e-06, 1.76373602e-06, 1.58273269e-06, 1.49813697e-06,
 1.32211395e-06, 1.44003524e-06, 1.42489429e-06, 1.10002716e-06,
 9.01008863e-07, 8.49881107e-07, 7.62521870e-07, 6.57641102e-07,
 5.85636641e-07, 5.33937361e-07, 4.16077216e-07, 3.33765858e-07,
 2.95575265e-07, 2.54744632e-07, 2.20144574e-07, 1.86314524e-07,
 1.77370967e-07, 1.54794344e-07, 1.47331687e-07, 1.39738552e-07,
 1.04110968e-07, 1.00786519e-07, 9.38635096e-08, 9.10853308e-08,
 8.71546329e-08, 7.48338889e-08, 6.06817433e-08, 5.66479201e-08,
 5.24576913e-08, 4.57020643e-08, 2.89942624e-08, 2.60449424e-08,
 2.17618741e-08, 2.10987991e-08, 1.75542296e-08, 1.34637033e-08,
 1.27167434e-08, 1.23258200e-08, 1.04987514e-08, 9.86368012e-09,
 8.49425727e-09, 9.33428117e-09, 7.42194387e-09, 6.46870902e-09,
 6.84633910e-09, 5.76456227e-09, 5.01137899e-09, 3.48686472e-09,
 2.91267189e-09, 2.77880628e-09, 1.73093447e-09, 1.42391210e-09,
 1.16455125e-09, 1.11815967e-09, 9.24976292e-10, 1.80003567e-10,
 1.97062420e-10, 2.61919378e-10, 6.95075432e-10, 6.13290805e-10,
 5.27572123e-10, 3.97019135e-15, 7.66958041e-16, -8.09588325e-16,
 3.86772503e-18, -4.22675086e-18, 0.00000000e+00, 0.00000000e+00,
 0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
 0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
 0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
 0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
 0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
 0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
 0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,

[illegible]

我们会看到一大堆值，但是其中的什么会引起我们的注意？我们会发现其中很多值都是 0。实际上，其中有超过 20% 的特征值都是 0。这就意味着这些特征都是其他特征的副本，也就是说，它们可以通过其他特征来表示，而本身并没有提供额外的信息。

接下来，我们了解一下部分数值的数量级。最前面 15 个值的数量级大于 10^5 ，实际上那以后的值都变得非常小。这就相当于告诉我们只有部分重要特征，重要特征的数目也很快就会下降。

最后，我们可能会注意到有一些小的负值，它们主要源自数值误差应该四舍五入成 0

在图9.4中已经给出了总方差的百分比，我们发现，在开始几个主成分之后，方差就会迅速下降。

```
dataMat = replaceNanWithMean()

#below is a quick hack copied from pca.pca()
meanVals = mean(dataMat, axis=0)
meanRemoved = dataMat - meanVals #remove mean
covMat = cov(meanRemoved, rowvar=0)
eigVals,eigVects = linalg.eig(mat(covMat))
eigValInd = argsort(eigVals)          #sort, sort goes smallest to largest
eigValInd = eigValInd[::-1]#reverse
sortedEigVals = eigVals[eigValInd]
total = sum(sortedEigVals)
varPercentage = sortedEigVals/total*100
```

```
fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(range(1, 21), varPercentage[:20], marker='^')
plt.xlabel('Principal Component Number')
plt.ylabel('Percentage of Variance')
plt.show()
```

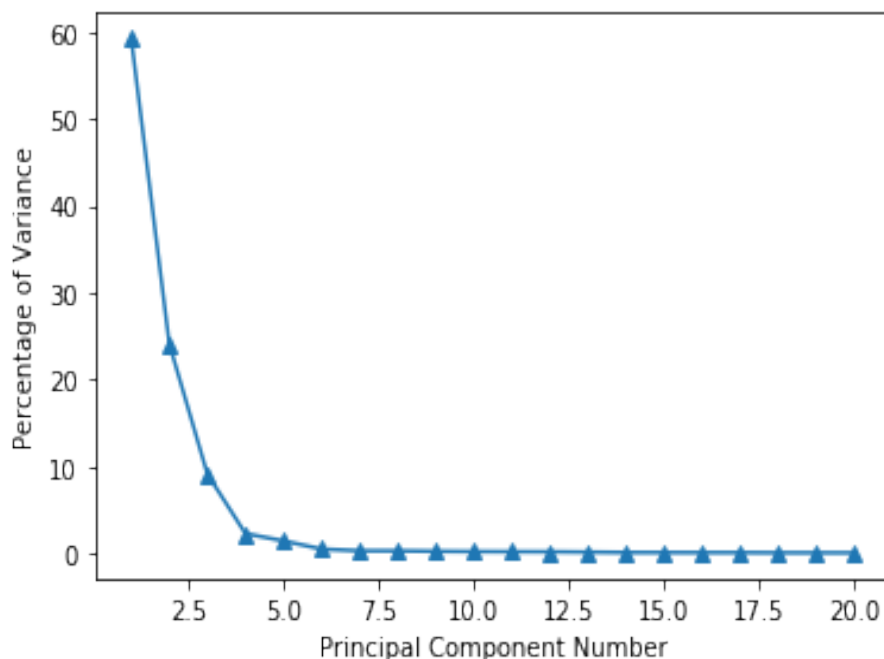


图 9.4: 前 20 个主成分占总方差的百分比。可以看出, 大部分方差都包含在前面的几个主成分中, 舍弃后面的主成分并不会损失太多的信息。如果保留前 6 个主成分, 则数据集可以从 590 个特征约简成 6 个特征, 大概实现了 1001 的压缩

如何选择主成分的个数。

我们可以尝试不同的截断值来检验它们的性能。有些人使用能包含 90% 信息量的主成分数量, 而其他人使用前 20 个主成分。我们无法精确知道所需要的主成分数目, 必须通过在实验中取不同的值来确定。有效的主成分数目则取决于数据集和具体应用。

9.3 PCA 的应用

9.3.1 使用 PCA 对高维数据可视化

通过对二维或者三维数据可视化可以轻松发现特征的分布。一个高维数据集无法用图进行可视化, 但我们依然可以通过将其减少到二维或者三维来洞察数据的结构。

Fisher 鸢尾花包括 3 种类别, 山鸢尾, 维吉尼亚鸢尾和变色鸢尾。一共有四个维度来表示特征。此数据集包含在 sklearn 类库中。

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

```

data = load_iris()
y = data.target
X = data.data
pca = PCA(n_components=2)
reduced_X = pca.fit_transform(X)

red_x, red_y = [], []
blue_x, blue_y = [], []
green_x, green_y = [], []
for i in range(len(reduced_X)):
    if y[i] == 0:
        red_x.append(reduced_X[i][0])
        red_y.append(reduced_X[i][1])
    elif y[i] == 1:
        blue_x.append(reduced_X[i][0])
        blue_y.append(reduced_X[i][1])
    else:
        green_x.append(reduced_X[i][0])
        green_y.append(reduced_X[i][1])
plt.scatter(red_x, red_y, c='r', marker='x')
plt.scatter(blue_x, blue_y, c='b', marker='D')
plt.scatter(green_x, green_y, c='g', marker='.')
plt.show()

```

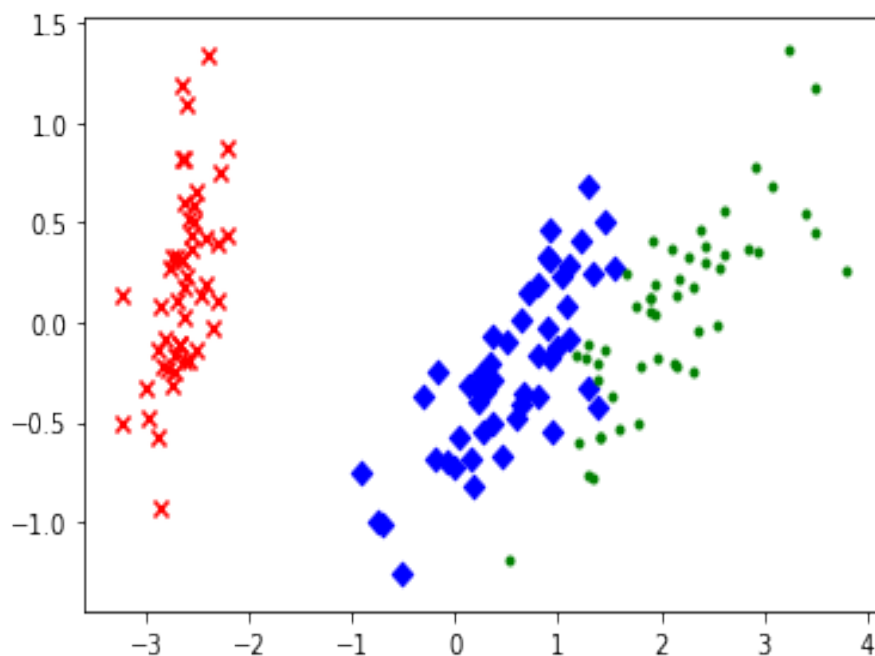


图 9.5: 利用 PCA 可视化鸢尾花数据集

9.3.2 利用 PCA 进行面部识别

第9章 练习

1. 面部识别问题是根据人的面部图像来识别一个人。

第一步：从[学在浙大](#)下载数据。

第二步：加载数据

第三步：降维

第四步：建立分类模型

第五步：性能度量（比较降维前后性能的差异）