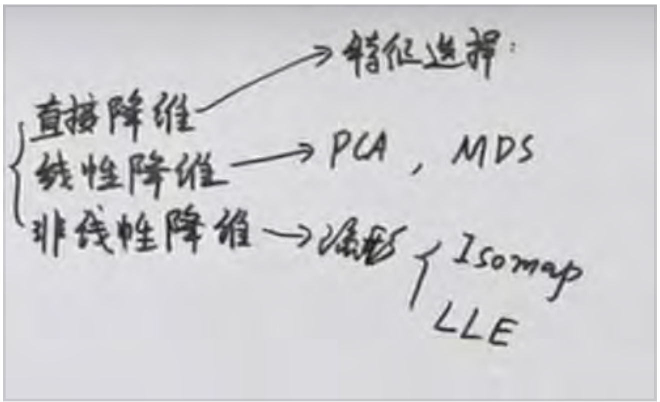
Week 13 笔记

PCA主成分分析（降维 dimension reduction）

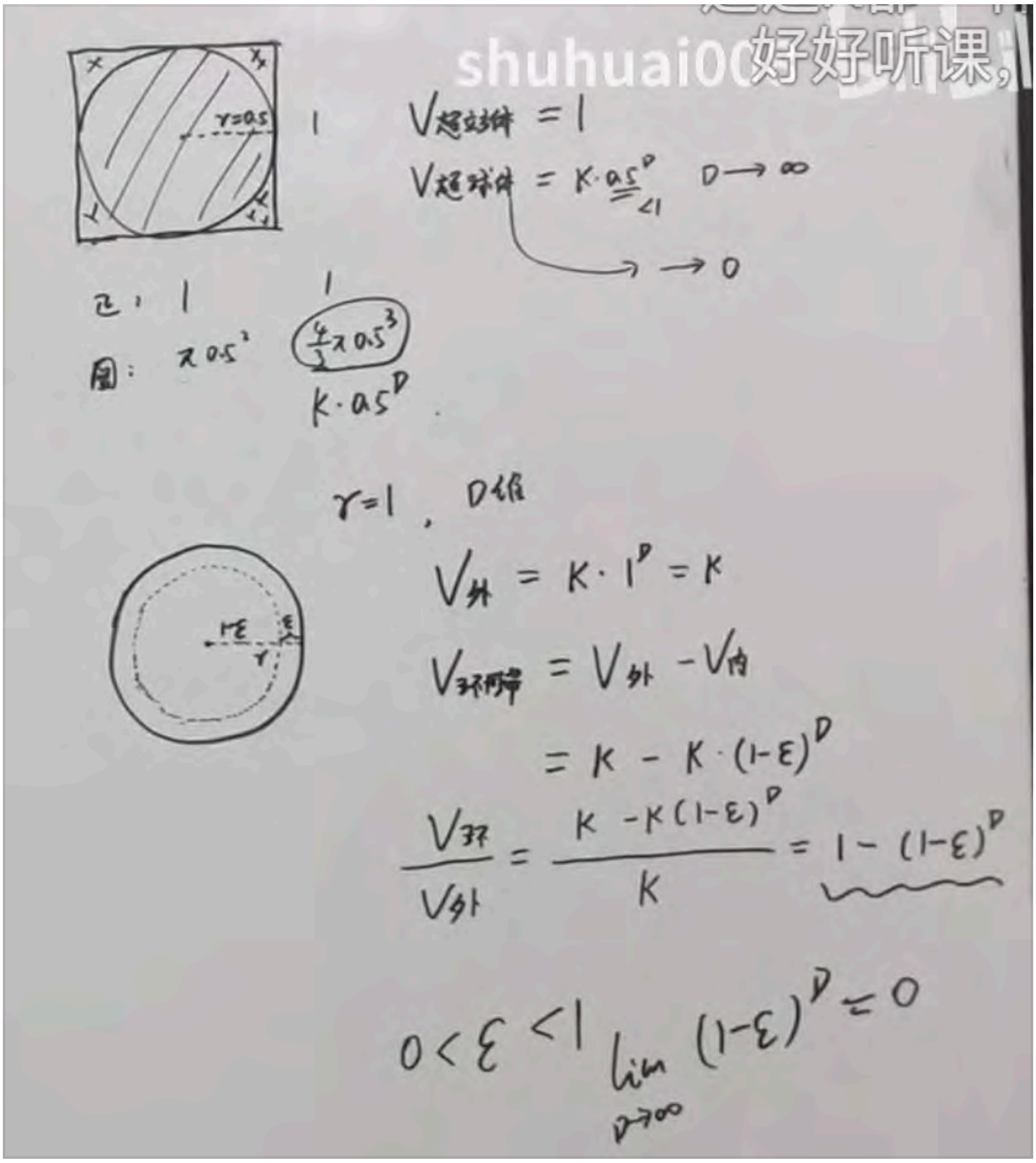
为什么要PCA？防止过拟合（提高模型泛化能力），方法有：

1. 正则化（将数据的relative size控制在同一数量级）
2. 增加样本量（增加训练样本）
3. 降低维度（防止维度灾难dimension curse）
    1. 降维方法：直接降维（直接选择期望的特征（特征降维）
    2. 线性降维：PCA MDS
    3. 非线性降维： 流形：Isomap LLE

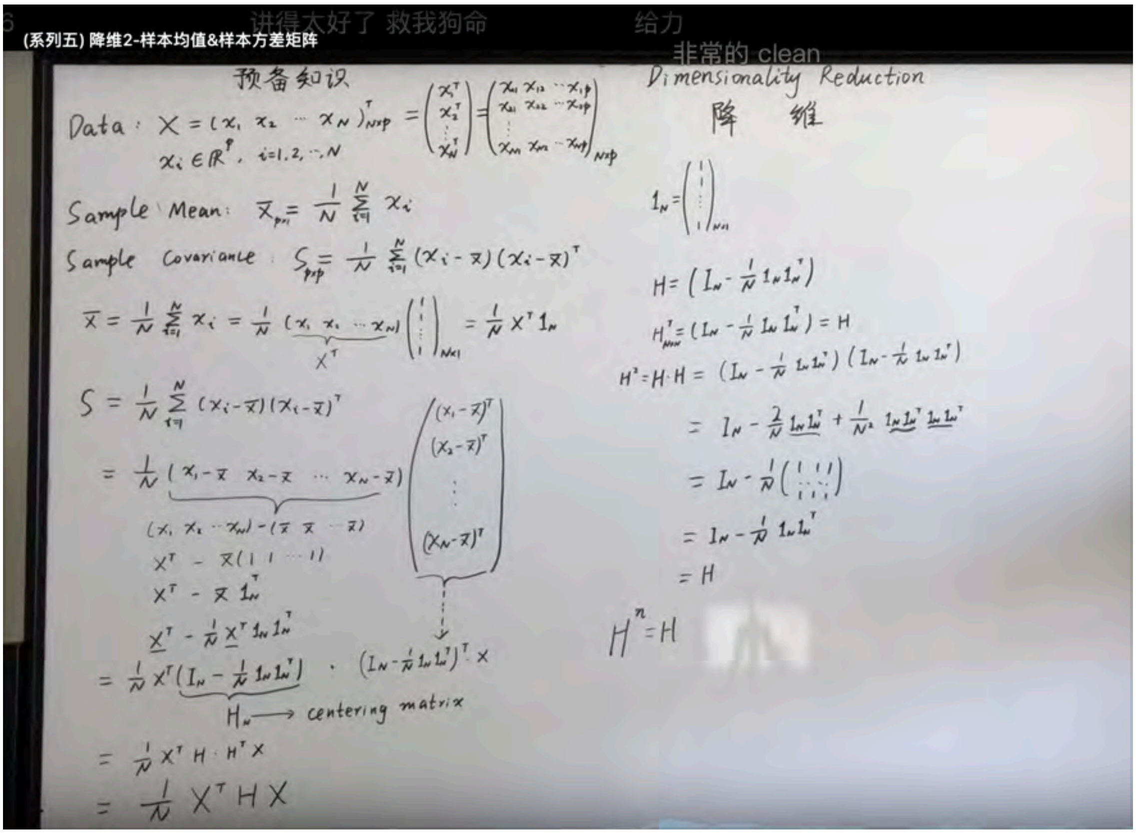


维度灾难：

1. 几何角度

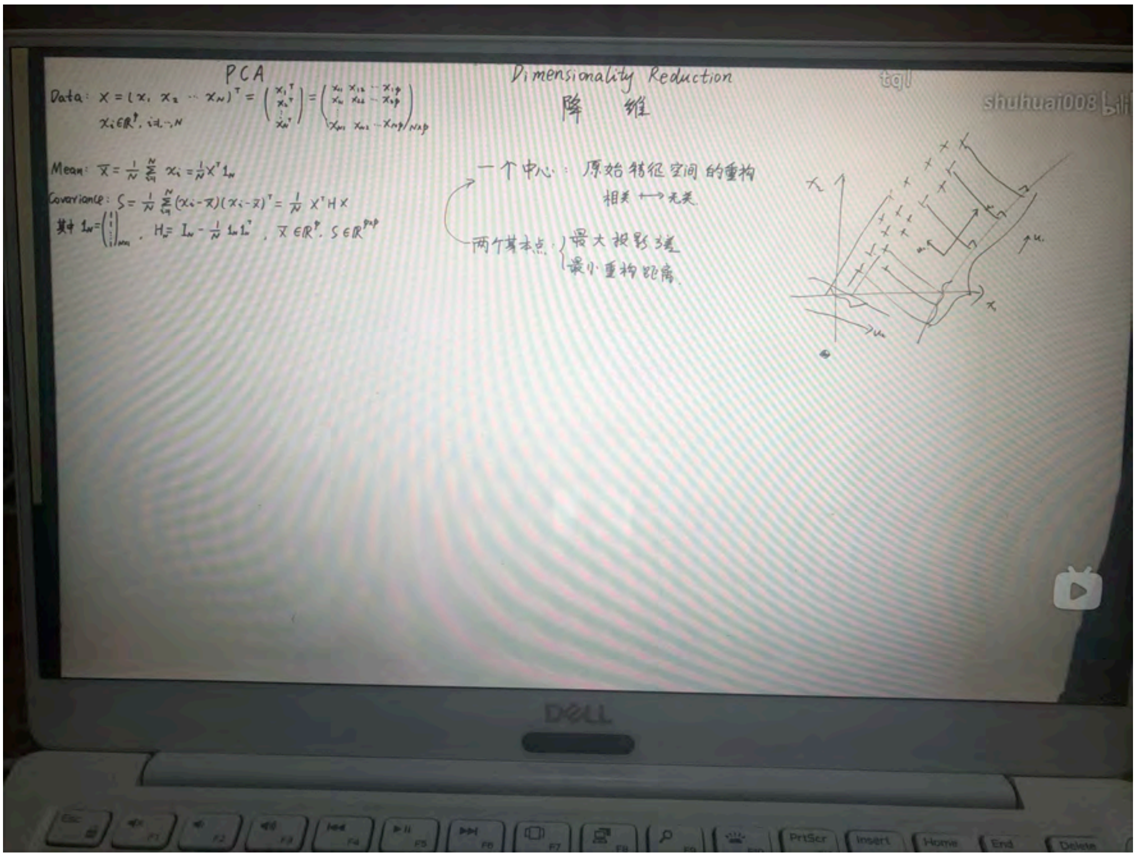


Lim 0->正无穷 (1-(1-epsilon)^D)=0
在高维空间球体相对体积变小 会导致数据稀疏性 且不均匀



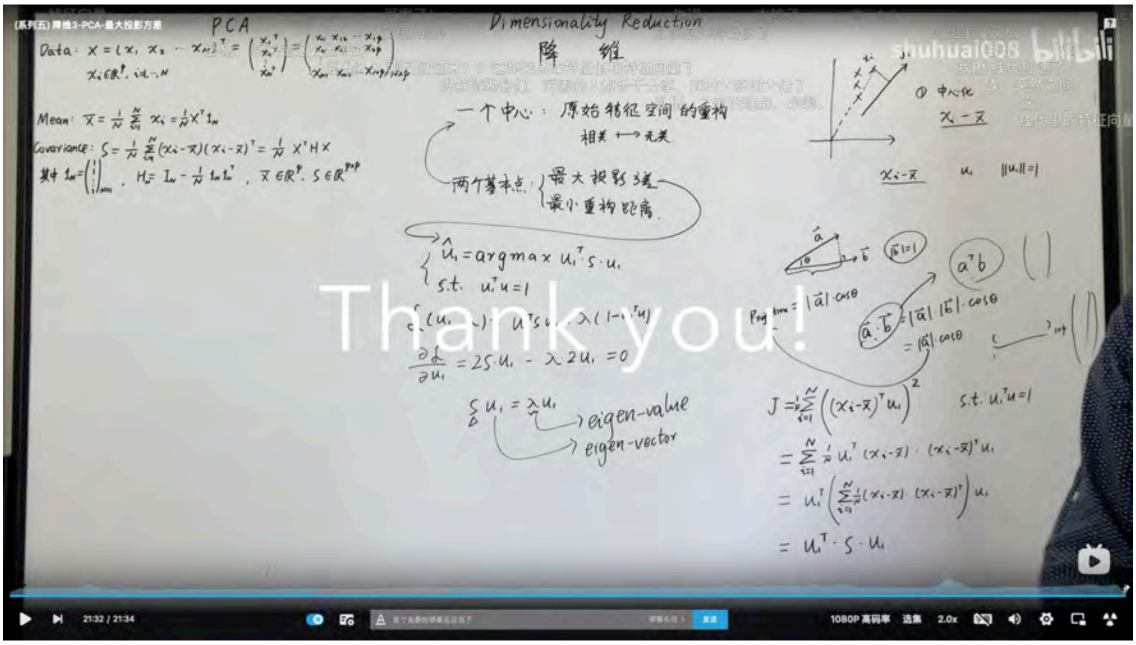
降维的矩阵表达

principal components analysis主成分分析

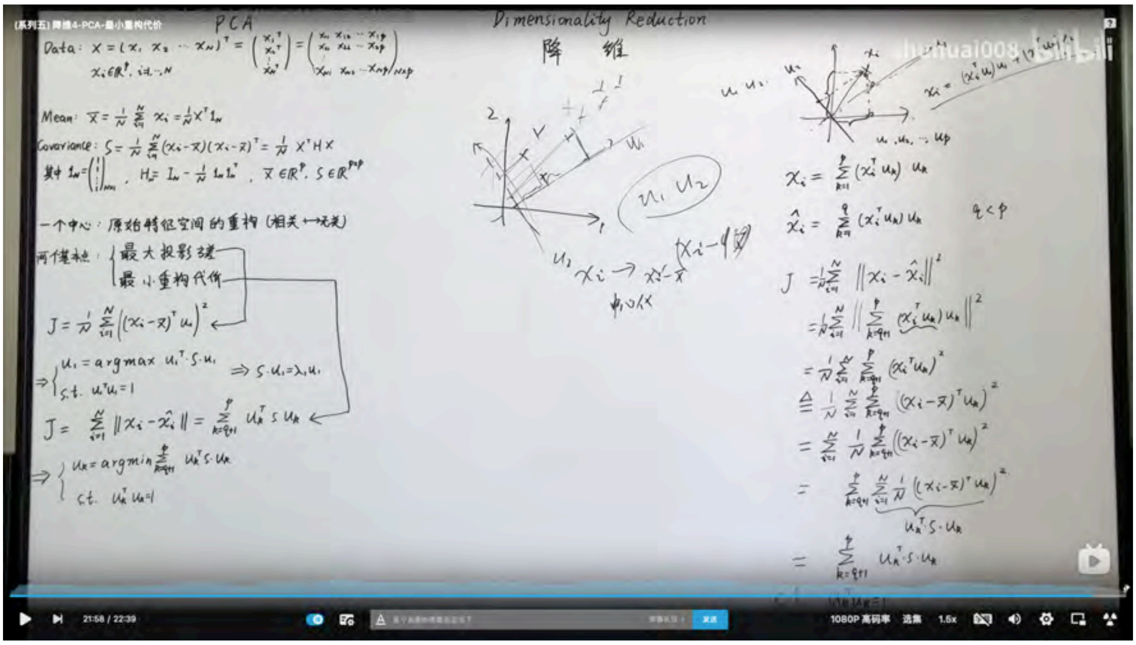


PCA降维方法 确保在降低维度后 在投影方向投影方差最大（即投影出来在一个维度上最稀疏）重构距离最小（在与投影维度正交的方向上最密集，实际上是对中心化

以后的数据进行重构）
先做平移，实现中心化
数学形式：
1. Max 投影方差：



2. 最小重构代价



滤波算法（中值，均值，卡尔曼滤波）

1. 中值滤波：

对于一个nxn的框体内的取中位数来替代原本放置在中心位置，matlab代码实现：

```
function [ img ] = median_filter( image, m )
%----------------------------------------------
% 中值滤波
% 输入：
```

```
%image：原图
%m：模板的大小3*3的模板，m=3

%输出：
%img：中值滤波处理后的图像
%--------------------------------------------------
    n = m;
    [ height, width ] = size(image);
    x1 = double(image);
    x2 = x1;
    for i = 1: height-n+1
        for j = 1:width-n+1
            mb = x1( i:(i+n-1), j:(j+n-1) );%获取图像中n*n的矩阵
            mb = mb(:);%将mb变成向量化，变成一个列向量
            mm = median(mb);%取中间值
            x2( i+(n-1)/2, j+(n-1)/2 ) = mm;

        end
    end

    img = uint8(x2);
end
```

2. 均值滤波：

均值滤波和中值滤波的区别在于取方框内的平均值来替代中位值，所以图片会更加平滑（锐度也更低）

3. 卡尔曼滤波：

卡尔曼滤波是用来解决现代控制问题的，假设有状态空间：

$$\mathbf{x}_t = \mathbf{F}_t \mathbf{x}_{t-1} + \mathbf{B}_t \mathbf{u}_t + \mathbf{w}_t,$$

和观测空间：

$$\mathbf{z}_t = \mathbf{H}_t \mathbf{x}_t + \mathbf{v}_t,$$

我们根据上一个时刻的Xt-1预测值和这一个时刻的观测值Zt，假定这两个值都满足高斯分布，且都是不完全准确的，核心在于找得到卡尔曼增益K，来对这两个值进行求加权平均：

$$S = HP'H^T + R$$

$$K = P'H^T S^{-1}$$

如果在简单一维问题下，就简化为了：

Time Update (prediction)	Measurement Update (correction)
$\hat{x}_k^- = \hat{x}_{k-1}$	$K_k = \frac{P_k^-}{P_k^- + R}$
$P_k^- = P_{k-1}$	$\hat{x}_k = \hat{x}_k^- + K_k(z_k - \hat{x}_k^-)$
	$P_k = (1 - K_k)P_k^-$

Pk是噪声的分布
（参考文献 https://ieeexplore.ieee.org/document/6279585
参考代码 https://zhuanlan.zhihu.com/p/45238681）

插值算法：主要用来在图片进行放大的时候怎么对于新的像素点来选择像素值，与采样互为逆运算

1. 最近邻插值：

```
def nearest(img, scale):
width, height, _ = img.shape
n_width = width*scale
n_height = height*scale
n_img =np.zeros((n_width, n_height, 3))
for k in range(3):
    for i in range(n_width):
        for j in range(n_height):
            #print(i, j, k)
            n_img[i,j, k] = img[round((i-1)/scale), round((j-1)/scale), k] #映射
    return Image.fromarray(np.uint8(n_img))
```

像素值映射公式为：

$$src_x = dst_x / scale$$

$$src_y = dst_y / scale$$

直接取最近的放缩后最近的一个像素点的值

2. 双线性插值

二维空间（坐标(x,y)，图片的像素值为我们关心的f(x)）中，我们需要计算出P点的像素值

取P点邻近的四个点Q11,Q12,Q21,Q22，并假设在邻近范围内，点的像素值是呈线性变化的。

这时我们先在x方向上进行线性插值，计算出R1,R2的像素值，

再在y方向上进行单次线性插值，求出P的像素值。

这里的这些点都是位于原始图像上的，我们只需要找到一个映射公式，将放大图像上的点与P点对应即可。

当然这里的映射公式就是最近邻插值中的映射公式。与最近邻插值不同的是，双线性插值法并没有之间将映射点的像素值作为放大图像的像素值，而是将映射点周围的四个点的加权作为放大图像的像素值。

$$f(R1) = \frac{x2-x}{x2-x1}f(Q11) + \frac{x-x1}{x2-x1}f(Q21)$$

$$f(R2) = \frac{x2-x}{x2-x1}f(Q12) + \frac{x-x1}{x2-x1}f(Q22)$$

$$f(P) = \frac{y2-y}{y2-y1}f(R1) + \frac{y-y1}{y2-y1}f(R2)$$

整合一下，就是：

$$f(P) = \frac{(x2-x)(y2-y)}{(x2-x1)(y2-y1)}f(Q11) + \frac{(x-x1)(y2-y)}{(x2-x1)(y2-y1)}f(Q21) + \frac{(x2-x)(y-y1)}{(x2-x1)(y2-y1)}f(Q12) + \frac{(x-x1)(y-y1)}{(x2-x1)(y2-y1)}f(Q22)$$

```
def double_linear(img, scale):
    width, height, _ = img.shape
    n_width = int(width*scale)
    n_height = int(height*scale)
    n_img = np.zeros((n_width, n_height, 3))
    for k in range(3):
        for i in range(n_width):
```

```
        for j in range(n_height):
            src_x = i/scale
            src_y = j/scale
            src_x_0 = int(np.floor(src_x))
            src_y_0 = int(np.floor(src_y))
            src_x_1 = min(src_x_0 + 1, width - 1)
            src_y_1 = min(src_y_0 + 1, height - 1)
            #print(src_x, src_y, src_x_0, src_y_0, src_x_1, src_y_1)
            value0 = (src_x_1 - src_x)*img[src_x_0, src_y_0, k] + (src_x - 
src_x_0)*img[src_x_1,src_y_0,k]
            value1 = (src_x_1 - src_x)*img[src_x_0, src_y_1, k] + (src_x - 
src_x_0)*img[src_x_1,src_y_1,k]
            n_img[i, j, k] = int((src_y_1-src_y) * value0 + (src_y-src_y_0)*value1)
    return Image.fromarray(np.uint8(n_img))
```

参考文献（最近邻插值、双线性插值与双三次插值 - ishihara的文章 - 知乎 https://zhuanlan.zhihu.com/p/428523385）

3. 拉格朗日插值

假定有3个点x_j,j=1,2,3，定义3个二次函数f_i, i=1,2,3，使得对应的二次函数在x_i处等于1，在其他两个点处等于0:

$$f_i(x_j), i = 1,2,3, j = 1,2,3$$

其中这三个二次函数需要满足:

$$f_i(x_j) = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

显然，我们构造的函数应该满足下列2式，第二个式子是更一般的场景:

$$f_1(x)=\frac{(x-x_2)(x-x_3)}{(x_1-x_2)(x_1-x_3)}$$

$$f_i(x)=\prod_{j\neq i}^{1\leq j\leq 3}\frac{(x-x_j)}{(x_i-x_j)}$$

最终我们可以得到正确的fitting曲线：

$$f(x)=\sum_{i=1}^{3}y_i f_i(x)$$