

# Unsupervised Learning

## Clustering 聚类

在这里没有y值存在，学习算法所做的就是根据间距等将一些没有标记的数据大致分成几类  
这样的算法比较适合市场细分，社交网络等等，没有明确名称的分类

### K-means algorithm K均值算法 最为广泛使用的聚类算法的一种

#### K-means algorithm

$$\mu_1 \quad \mu_2$$

Randomly initialize  $K$  cluster centroids  $\mu_1, \mu_2, \dots, \mu_K \in \mathbb{R}^n$

Repeat {

*Cluster assignment step*

for  $i = 1$  to  $m$   
 $c^{(i)} :=$  index (from 1 to  $K$ ) of cluster centroid  
closest to  $x^{(i)}$

min  $\|x^{(i)} - \mu_k\|^2$

for  $k = 1$  to  $K$   
 $\rightarrow \mu_k :=$  average (mean) of points assigned to cluster  $k$   
 $x^{(1)}, x^{(5)}, x^{(6)}, x^{(10)}$        $\rightarrow c^{(1)}=2, c^{(5)}=2, c^{(6)}=2, c^{(10)}=2$

*Move centroid*

$\mu_2 = \frac{1}{4} [x^{(1)} + x^{(5)} + x^{(6)} + x^{(10)}] \in \mathbb{R}^n$

上图中可以看见，K均值算法分为两步，

前提：随机初始化K的聚类中心（cluster centroid），总共有k个

第一步，簇分配，遍历所有样本，根据每个样本距离聚类中心的距离将样本分为k组；

第二步，移动聚类中心，求每一组数据的均值，然后将聚类中心移动到均值点上。

重复以上两步，最终聚类中心会移动到一个平衡点并不再移动，最终我们将得到k组分类

## K-means optimization objective

- $c^{(i)}$  = index of cluster (1, 2, ..., K) to which example  $x^{(i)}$  is currently assigned
- $\mu_k$  = cluster centroid  $k$  ( $\mu_k \in \mathbb{R}^n$ )  $K$   $k \in \{1, 2, \dots, K\}$
- $\mu_{c(i)}$  = cluster centroid of cluster to which example  $x^{(i)}$  has been assigned  $x^{(i)} \rightarrow S$   $c^{(i)} = 5$   $\underline{\mu_{c(i)}} = \underline{\mu_5}$

Optimization objective:

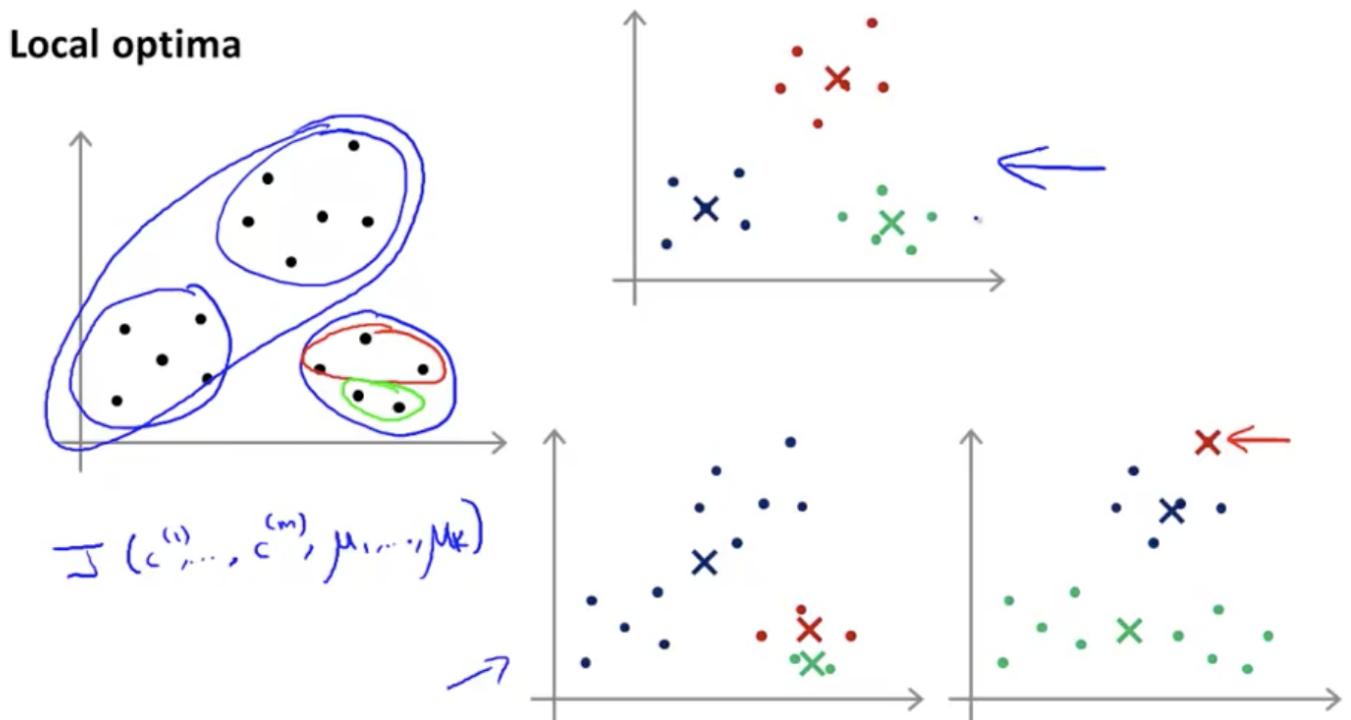
$$\rightarrow J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K) = \frac{1}{m} \sum_{i=1}^m \|x^{(i)} - \mu_{c(i)}\|^2$$

$\min_{c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K} J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K)$  *Distortion*

上图中  $c(i)$  表示当前样本  $x(i)$  所归为的簇的序号，  $K$  表示簇的总数，  $u_k$  表示第  $k$  个聚类中心，  $u_{c(i)}$  表示  $x(i)$  所属的那个簇的聚类中心，  $c(i) = 5$  表示  $x(i)$  被划分到了第 5 个样本这样我们就可以知道 K 均值函数的优化目标了，也就是关于  $c$  和  $u$  的代价函数  $J$  (也叫失真函数) 在第一步簇分配过程中，保持  $u$  不变，从  $c(1)$  到  $c(m)$ ，选择与  $x$  距离最小的  $c$ ，在第二步移动聚类中心的过程中，保持  $c$  不变，从  $u_1$  到  $u_K$  而找到距离所有  $x$  最小的  $u$ ，最后反复循环以上两步，最终我们可以得到  $J$

## K的初始化

最流行最有效的方法是随机选取若干个样本并将其设定为聚类中心，然后重复几次，以避免局部最优解，例如下图中的情况



值得注意的是，当你的聚类数相当多，有成百上千个，那么多次随机初始化不会有很好的效果，当K在2到10之间，多次的随机初始化通常能保证一个较好的局部最优解

## 聚类数量的选择

结论：没有一个非常完美的办法，最好的方法是根据实际的业务需要

可行方法之一：手肘法

当你的曲线有一个明显的拐点，例如下图左边的K=3的情况，

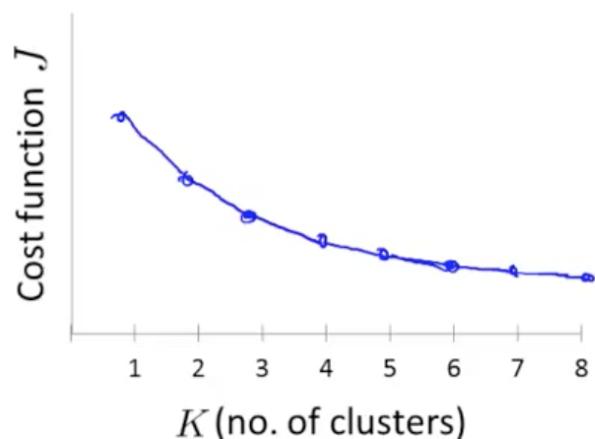
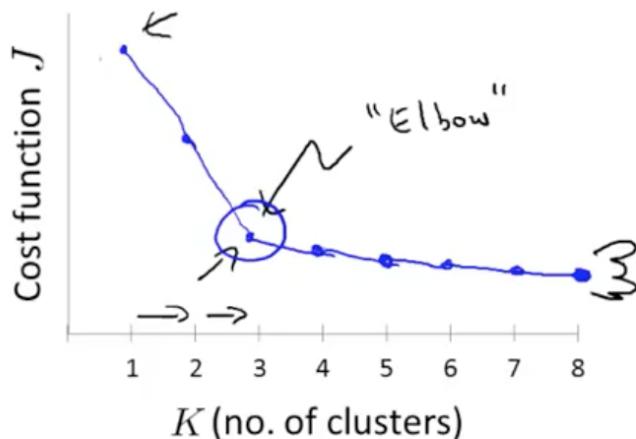
这时你可以用拐点的数值最为聚类中心的数量，

这个方法不值得过于期待，因为大部分时间我们的曲线长成下图右边这个样子，

下降平缓而没有一个明显的拐点

### Choosing the value of K

Elbow method:



另一方面，根据实际情况来选择聚类数量显然更为明智。

例如下图中衬衫分类的栗子，分为3类可以降低成本，分为5类可以更好地适应顾客，

根据你的business的需要和业务观点来进行选择是目前的主流方式。

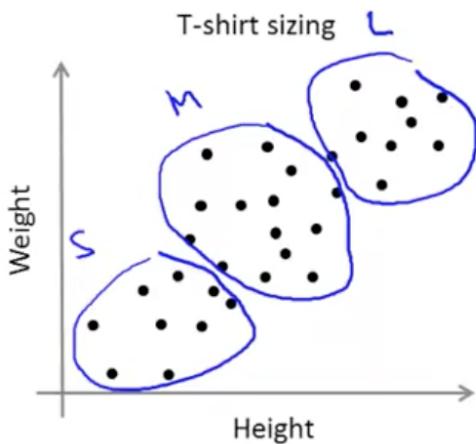
因此，做之前先搞清楚你的目的是什么。

## Choosing the value of K

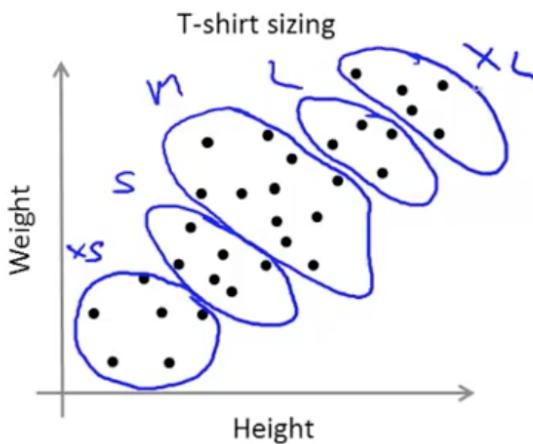
Sometimes, you're running K-means to get clusters to use for some later/downstream purpose. Evaluate K-means based on a metric for how well it performs for that later purpose.

$K=3 \quad S, M, L$

E.g.



$K=5 \quad XS, S, M, L, XL$



## Dimensionality Reduction 维数约减

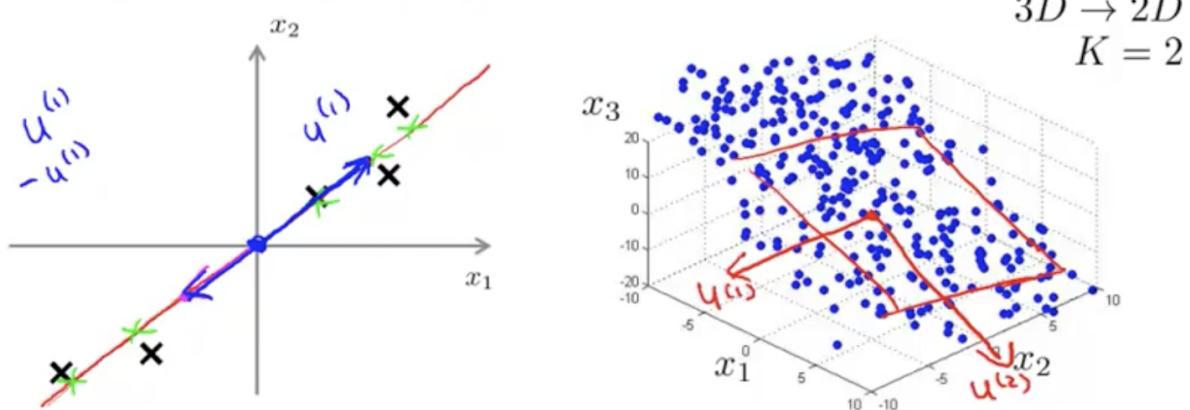
### 无监督学习的第二种方法

维数约减便于我们压缩数据以及将算法可视化，因为能用图形表现的数据不会超过三维  
对于降维问题来说，现在最流行的算法是PCA

这个方法之所以成立，是因为大部分现实中的数据，许多特征变量都是高度相关的，  
所以即使大量压缩数据仍然会保留99%的差异性

## Principal Component Analysis (PCA) 主成分分析法

## Principal Component Analysis (PCA) problem formulation



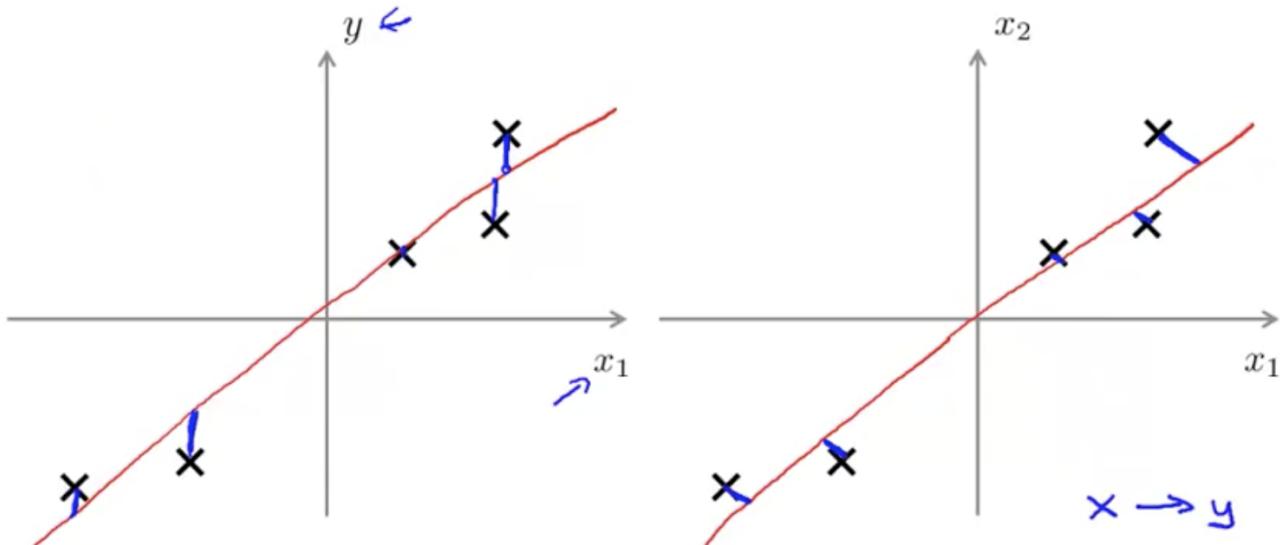
Reduce from 2-dimension to 1-dimension: Find a direction (a vector  $\underline{u^{(1)} \in \mathbb{R}^n}$ ) onto which to project the data so as to minimize the projection error.

Reduce from n-dimension to k-dimension: Find  $k$  vectors  $\underline{u^{(1)}, u^{(2)}, \dots, u^{(k)}}$  onto which to project the data, so as to minimize the projection error.

要从n维降到k维，找出k个向量使多维数据投射到该向量的距离最小

例如在3维数据中，找出 $u_1$ 和 $u_2$ 两个向量，定义一个能够投射所有点的二维平面  
或者专业点说，将数据投影到这k个向量展开的线性子空间上

## PCA is not linear regression



PCA不是线性回归，他们计算的数据的误差形式不同（PCA与向量垂直，线性回归与坐标轴垂直）

并且PCA中没有一个y来给出衡量标准（坐标轴不同）

尽管他们看上去很相似，但确实两个完全不同的算法

最后需要注意的是，在进行PCA之前需要进行均值归一（mean normalization）和特征缩放（feature scaling）（例如规范到-1~1之间）

## Data preprocessing

Training set:  $x^{(1)}, x^{(2)}, \dots, x^{(m)}$  ←

Preprocessing (feature scaling/mean normalization):

$$\mu_j = \frac{1}{m} \sum_{i=1}^m x_j^{(i)}$$

Replace each  $x_j^{(i)}$  with  $x_j^{(i)} - \mu_j$ .

If different features on different scales (e.g.,  $x_1$  = size of house,  $x_2$  = number of bedrooms), scale features to have comparable range of values.

$$x_j^{(i)} \leftarrow \frac{x_j^{(i)} - \mu_j}{s_j}$$

我们可以用 $x^{(i)j}$ 减去 $\mu_j$ 除以 $s_j$ 来替换掉 $x^{(i)j}$ , 这里 $s_j$ 代表特征j的某个度量范围, 因此可以表示最大值减最小值,

或者更普遍的, 它可以表示特征j的标准差

## Principal Component Analysis (PCA) algorithm

Reduce data from  $n$ -dimensions to  $k$ -dimensions

Compute "covariance matrix":

$$\Sigma = \frac{1}{m} \sum_{i=1}^n (x^{(i)}) (x^{(i)})^T$$

Compute "eigenvectors" of matrix  $\Sigma$ :

$$\rightarrow [U, S, V] = \text{svd}(\Sigma); \quad \rightarrow \text{Singular value decomposition}$$

eig ( $\Sigma$ )

$n \times n$  matrix.

$$U = \begin{bmatrix} | & | & | & \dots & | \\ u^{(1)} & u^{(2)} & u^{(3)} & \dots & u^{(n)} \\ | & | & | & \dots & | \end{bmatrix} \quad U \in \mathbb{R}^{n \times n}$$

$U^{(1)}, \dots, U^{(k)}$

把数据从 $n$ 维降到 $k$ 维, 首先要做的是计算出协方差矩阵, 用希腊字母大写的西格玛 $\Sigma$ 来表示 (注意不要和求和符号混淆)

计算出协方差矩阵后, 我们要接着求它的特征向量 (eigenvectors)

在Octave中, svd(sigma)可以实现这一功能, svd表示奇异值分解 (singular value decomposition)

在Octave中, 还有一个命令eig, 能够和svd得出相同的结果, 不过svd更为稳定一些

我们能得到相同的答案是因为协方差矩阵总满足一个数学性质称为对称正定 (symmetric positive definite) (这里不用深究)

svd将会输出三个矩阵, 其中我们真正需要的是 $n \times n$ 维矩阵U

然后我们取矩阵U的前k列，得到一个n\*k维的矩阵 (Ureduce)，见下图

$$x \in \mathbb{R}^n \rightarrow z \in \mathbb{R}^k$$

$$z = \begin{bmatrix} u^{(1)} & u^{(2)} & \dots & u^{(k)} \end{bmatrix}^T \quad U_{\text{reduce}}$$

$$x = \begin{bmatrix} (u^{(1)})^T \\ \vdots \\ (u^{(k)})^T \end{bmatrix} \quad k \times n \quad k \times 1$$

得出n\*k维的向量 (Ureduce) 后将其转置，再乘以x，就可以得到一个k\*1维的向量z  
这里的x可以是训练集也可以是交叉检验集

## Principal Component Analysis (PCA) algorithm summary

→ After mean normalization (ensure every feature has zero mean) and optionally feature scaling:

$$\Sigma = \frac{1}{m} \sum_{i=1}^m (x^{(i)})(x^{(i)})^T$$

$$\rightarrow [U, S, V] = \text{svd}(\Sigma);$$

$$\rightarrow U_{\text{reduce}} = U(:, 1:k);$$

$$\rightarrow z = U_{\text{reduce}}' * x;$$

$x \in \mathbb{R}^n \quad x_0 \neq 1$

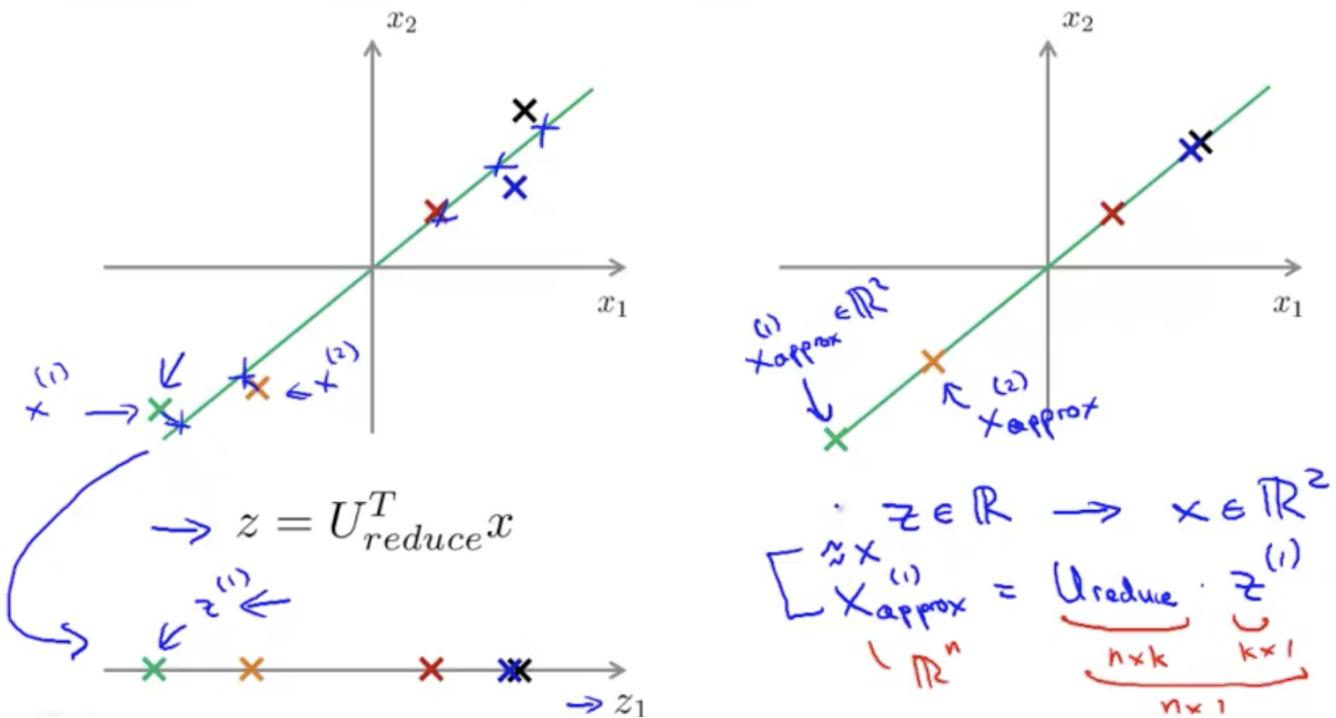
$$\Sigma = \begin{bmatrix} x^{(1)\top} \\ \vdots \\ x^{(m)\top} \end{bmatrix}$$

$$\rightarrow \Sigma = (1/m) * X' * X;$$

计算小技巧，如果X是上图那样的矩阵，那么我们可以用 $X' * X$ 来得到和左边sum之后相同的结果

维度可以约减，也可以还原，如下图

## Reconstruction from compressed representation



$X_{approx}(1) = U_{reduce} * z(1)$  可以得到还原后的  $x(1)$ , 与真正的  $x$  还是有点差距因为还原后的结果已经是投射到向量上的结果了

## PCA应用篇

### 选择主成分数量k

当我们将向量  $x$  从  $n$  维降到  $k$  维, 这里的就是主成分的数量。

那么现在实际应用中, 我们要怎么决定  $k$  的取值呢, 见下图

### Choosing $k$ (number of principal components)

Average squared projection error:  $\frac{1}{m} \sum_{i=1}^m \|x^{(i)} - x_{approx}^{(i)}\|^2$

Total variation in the data:  $\frac{1}{m} \sum_{i=1}^m \|x^{(i)}\|^2$

Typically, choose  $k$  to be smallest value so that

$$\rightarrow \frac{\frac{1}{m} \sum_{i=1}^m \|x^{(i)} - x_{approx}^{(i)}\|^2}{\frac{1}{m} \sum_{i=1}^m \|x^{(i)}\|^2} \leq 0.01 \quad (1\%)$$

“99% of variance is retained”

上图中涉及到的几个概念如下：

平均平方映射误差 (Average Squared Projection Error)

PCA就是要将这个量最小化

$x_{approx}(i)$ 几映射值

数据的总变差 (Total Variation)

上图中的分母，是这些样本 $x(i)$ 的长度的平方的均值，他的意思是「平均来看，我们的训练样本距离零向量（原点）有多远？」

在这里，一个常见的选择K值的经验法则是，选择能够使得它们之间的比例小于等于0.01的最小的k值。

换言之一个非常常用的选择k值的方法是，我们希望平均平方映射误差，也就是 $x$ 和其映射值之间的平均距离

除以数据的总变差能够小于一个我们希望的比例，

例如当k使得平均平方映射误差小于等于1%时，它的意思是，保留99%的差异性的最小的k值

在实际应用中，如果我们每取一个k就遍历所有数据去计算新向量 $z$ 和映射值 $x_{approx}$ ，那会非常没有效率，

事实上，我们可以利用PCA法中用到的svd返回来的三个矩阵中的矩阵S来计算平均平方映射误差，见下图

## Choosing $k$ (number of principal components)

Algorithm:

Try PCA with  $k = 1, 2, 3, \dots, n$

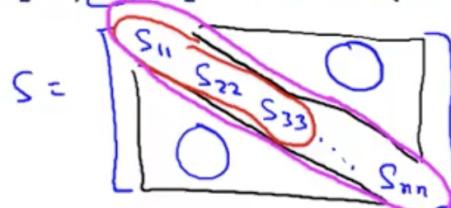
Compute  $U_{reduce}, z^{(1)}, z^{(2)}, \dots, z^{(m)}, x_{approx}^{(1)}, \dots, x_{approx}^{(m)}$

Check if

$$\frac{\frac{1}{m} \sum_{i=1}^m \|x^{(i)} - x_{approx}^{(i)}\|^2}{\frac{1}{m} \sum_{i=1}^m \|x^{(i)}\|^2} \leq 0.01?$$

$k = 17$

$$\rightarrow [U, S, V] = svd(Sigma)$$



For given  $k$

$$1 - \frac{\sum_{i=1}^k S_{ii}}{\sum_{i=1}^n S_{ii}} \leq 0.01$$

$$\frac{\sum_{i=1}^k S_{ii}}{\sum_{i=1}^n S_{ii}} \geq 0.99$$

S是一个对角线上元素是 $s_{11}, s_{22}$ , 一直到 $s_{nn}$ 的非零元素，而对角以外都是0的矩阵

举个栗子，如果我们要把维数约减到 $k$ 并计算平均平方映射误差，那么我们只要计算从 $i=1$ 到 $k$ 对 $S_{ii}$ 求和（就是对角线这些元素的总和），然后除以从 $i=1$ 到 $n$ 对 $S_{ii}$ 求和，最后用1减去这个值，就等价于左边蓝框中的数值

最后简化一下，就是人们常用的求 $k$ 值的公式，如下图：

## Choosing $k$ (number of principal components)

→  $[U, S, V] = \text{svd}(\Sigma)$

Pick smallest value of  $k$  for which

$$\frac{\sum_{i=1}^k S_{ii}}{\sum_{i=1}^m S_{ii}} \geq 0.99$$

$k=100$

(99% of variance retained)

## 关于PCA使用的一些技巧

### Supervised learning speedup

→  $(\underline{x}^{(1)}, y^{(1)}), (\underline{x}^{(2)}, y^{(2)}), \dots, (\underline{x}^{(m)}, y^{(m)})$

Extract inputs:

Unlabeled dataset:  $\underline{x}^{(1)}, \underline{x}^{(2)}, \dots, \underline{x}^{(m)} \in \mathbb{R}^{10000}$  ←  
↓ PCA

$x^{(i)} \in \mathbb{R}^{10000}$  ←  
100  
100

New training set:

$(\underline{z}^{(1)}, y^{(1)}), (\underline{z}^{(2)}, y^{(2)}), \dots, (\underline{z}^{(m)}, y^{(m)})$        $h_\theta(z) = \frac{1}{1 + e^{-\theta^\top z}}$

Note: Mapping  $x^{(i)} \rightarrow z^{(i)}$  should be defined by running PCA only on the training set. This mapping can be applied as well to the examples  $x_{cv}^{(i)}$  and  $x_{test}^{(i)}$  in the cross validation and test sets.

上图中，在使用监督学习时我们也可以用PCA来对x进行降维处理，以达到提升算法速度的目的。需要注意的是，这种从x到z的对应关系只可以通过在训练集上运行PCA定义出来。

具体来讲，这种PCA所学习出的对应关系所做的就是计算出一系列的参数，这就是特征缩放和均值归一化。

同时也计算出这样一个降维的矩阵Ureduce，但是降维矩阵Ureduce中的数据就像一个PCA所学习的参数一样。

我们需要使我们的参数唯一地适应这些训练集，而不是适应我们的交叉验证或者测试集。

总结一下，当我们在运行PCA的时候，只是在训练集那一部分来进行的，而不是交叉验证的数据集，这就定义了从x到z的映射，

然后你就可以将这个映射应用到交叉验证数据集中和测试数据集中。

如上面提到的，PCA的主要优点是数据压缩和可视化。

此外，关于PCA有一个不推荐的应用，就是用PCA来避免过拟合。

虽然这样做可能会得到一个还不错的结果，但是完全不如正规化来得有效，

一个重要的原因是，PCA的副作用在于即使保留了99%的差异性，PCA仍有可能舍弃一些关键特征，

因为我们的主要目的是压缩数据以加速算法，这样的损失就是不可避免的，

但是当有正规化这种完全可以避免损失的方法存在时，再用PCA去避免过拟合就是一个不明智的选择了。

另一方面，也正是由于「舍弃重要特征的可能性」这一缺点，当我们设计一个学习算法时，

应该避免一开始就把使用PCA考虑在内，而是先用原始数据来构建算法，

当结果不是很理想，例如收敛缓慢或者占用了大量内存时，才考虑使用PCA