数据预处理

1. 在获取的数据中经常会遇到唯一属性，这些属性通常是人为添加的一些`id`属性，如存放在数据库中的自增的主键。

   对于这些属性，它并不能刻画样本自身的分布规律。所以只需要简单的删除这些属性。

2. 对于数据中的某个属性，如果其方差很小，则意味着其识别能力较弱。极端情况下其方差为0，这意味着该属性在所有样本上的值都是恒定的。

   因此可以设定一个阈值（如 10^{-3} ），将方差小于该阈值的属性直接剔除。

一、缺失值处理

1. 数据缺失值的产生的原因多种多样，主要分为客观原因和人为原因。
   - 客观原因：比如数据存储的失败、存储器损坏、机械故障导致某段时间数据未能收集（对于定时数据采集而言）。
   - 人为原因：由于人的主观失误（如：数据录入人员失误漏录了数据）、历史局限（如：数据在早期尚无记录）、有意隐瞒数据（如：在市场调查中被访人拒绝透露相关问题的答案，或者回答的问题是无效的）导致数据未能收集。
2. 缺失值的处理有三种方法：
   - 直接使用含有缺失值的数据。

     某些算法可以直接使用含有缺失值的情况，如决策树算法可以直接使用含有缺失值的数据。
     - 优点：直接使用原始数据，排除了人工处理缺失值带来的信息损失。
     - 缺点：只有少量的算法支持这种方式。
   - 删除含有缺失值的数据。

     最简单的办法就是删除含有缺失值的样本。
     - 优点：简单、高效。
     - 缺点：如果样本中的缺失值较少，则直接丢弃样本会损失大量的有效信息。这是对信息的极大浪费。

       > 如果样本中包含大量的缺失值，只有少量的有效值，则该方法可以接受。
   - 缺失值补全。

     用最可能的值来插补缺失值。这也是在实际工程中应用最广泛的技术。
     - 优点：保留了原始数据
     - 缺点：计算复杂，而且当插补的值估计不准确时，会对后续的模型引入额外的误差。
3. 缺失值补全常见有以下方法：
   - 均值插补
   - 同类均值插补
   - 建模预测
   - 高维映射
   - 多重插补
   - 压缩感知及矩阵补全

1.1 均值插补 && 同类均值插补

1. 均值插补：
   - 如果样本的属性是连续值，则该属性的缺失值就以该属性有效值的平均值来插补。
   - 如果样本的属性是离散值，则该属性的缺失值就以该属性有效值的众数（出现频率最高的值）来插补。
2. 均值插补在含有缺失值的属性上的所有缺失值都填补为同一个值。

   而同类均值插补首先将样本进行分类，然后以该类中的样本的均值来插补缺失值。

1.2 建模预测

1. 建模预测的思想是：将缺失的属性作为预测目标，通过建立模型来预测。
2. 给定数据集 $\mathbb{D}=\{(\vec{\mathbf{x}}_1,\tilde{y}_1),(\vec{\mathbf{x}}_2,\tilde{y}_2),\cdots,(\vec{\mathbf{x}}_N,\tilde{y}_N)\}$。

   假设属性 j 含有缺失值，根据 $x_{i,j}$ 是否缺失，将数据集划分为：
   - $\mathbb{D}_1=\{\vec{\mathbf{x}}_i\mid x_{i,j}\neq null\}$ ： 属性 j 有效的样本的集合。
   - $\mathbb{D}_2=\{\vec{\mathbf{x}}_i\mid x_{i,j}= null\}$ ： 属性 j 缺失的样本的集合。

   将 $\mathbb{D}_1$ 中的样本作为新的训练集，标签值重新定义为属性 j 的值，通过建模来完成属性 j 的学习。将 $\mathbb{D}_2$ 中的样本作为测试集，通过学得的模型来预测其属性 j 的值。
3. 这种方法的效果相对较好，但是该方法有个根本缺陷：
   - 如果其他属性和属性 j 无关，则预测的结果无意义。
   - 如果预测结果相当准确，则又说明属性 j 可以由其它属性计算得到， 于是属性 j 信息冗余，没有必要纳入数据集中。

   一般的情况是介于两者之间。

1.3 高维映射

1. 高维映射的思想是：将属性映射到高维空间。
2. 给定数据集 $\mathbb{D}$ ，假设属性 j 的取值为离散值 $\{a_1,a_2\cdots,a_K\}$ 一共 K 个值，则将该属性扩展为 $K+1$ 个属性 $(j_1,j_2,\cdots,j_{K+1})$ ，其中：
   - 若样本在属性 j 上的取值为 a_k，则样本在新的属性 j_k 上的取值为 1，在新的属性 $j_1,\cdots,j_{k-1},j_{k+1},\cdots,j_{K+1}$ 上的取值为 0 。
   - 若样本在属性 j 上缺失，则样本的新的属性 j_{K+1} 上的取值为 1,在新的属性 $j_1,\cdots,j_K$ 上取值为 0 。
3. 对于连续特征，高维映射无法直接处理。可以在连续特征离散化之后，再进行高维映射。
4. 高维映射是最精确的做法，它完全保留了所有的信息，也未增加任何额外的信息。比如广告的`CTR`预估模型，预处理时会把所有变量都这样处理，达到几亿维。
   - 优点：完整保留了原始数据的全部信息。
   - 缺点：计算量大大提升。而且只有在样本量非常大的时候效果才好，否则会因为过于稀疏，效果很差。

1.4 多重插补

1. 多重插补（`Multiple Imputation:MI`）认为待插补的值是随机的，它的值来自于已观测到的值。

   具体实践上通常是估计出待插补的值，然后再加上不同的噪声，形成多组可选插补值。然后根据某种选择依据，选取最合适的插补值。
2. 多重插补法的步骤：
   - 通过变量之间的关系对缺失数据进行预测，利用蒙特卡洛方法生成多个完整的数据集。

- 在每个完整的数据集上进行训练，得到训练后的模型以及评价函数值。
- 对来自各个完整的数据集的结果，根据评价函数值进行选择，选择评价函数值最大的模型，其对应的插值就是最终的插补值。

1.5 压缩感知 && 矩阵补全

1. 在现实任务中，经常希望根据部分信息来恢复全部信息。压缩感知和矩阵补全就是用于完成这个任务。
2. 假定有长度为 n 的离散信号 $\vec{\mathbf{x}}$ 。 根据奈奎斯特采样定理，当采样频率达到 $\vec{\mathbf{x}}$ 最高频率的两倍时，采样后的信号就保留了原信号的全部信息。

   假定以远小于奈奎斯特采样定理要求的采样频率进行采样，得到了长度为 m 的采样后信号 $\vec{\mathbf{y}}$， 其中 $m \ll n$ 。 则有：$\vec{\mathbf{y}} = \mathbf{\Phi}\vec{\mathbf{x}}$ 。

   其中 $\mathbf{\Phi} \in \mathbb{R}^{m\times n}$ 是对信号 $\vec{\mathbf{x}}$ 的测量矩阵，它确定了以什么样的频率采样以及如何将采样样本组成采样后的信号。
3. 通常在已知离散信号 $\vec{\mathbf{x}}$ 和测量矩阵 $\mathbf{\Phi}$ 时要得到测量值 $\vec{\mathbf{y}}$ 很容易。但是如果给定测量值 $\vec{\mathbf{y}}$ 和测量矩阵 $\mathbf{\Phi}$， 要还原出原始信号 $\vec{\mathbf{x}}$ 比较困难。这是由于当 $m \ll n$ 时， $\vec{\mathbf{y}} = \mathbf{\Phi}\vec{\mathbf{x}}$ 是一个欠定方程，无法简单的求出数值解。

   假设存在某种线性变换 $\mathbf{\Psi} \in \mathbb{R}^{n\times n}$ ， 使得 $\vec{\mathbf{x}} = \mathbf{\Psi}\vec{\mathbf{s}}$，其中 $\vec{\mathbf{s}}$ 也和 $\vec{\mathbf{x}}$ 一样是 n 维列向量，则有 $\vec{\mathbf{y}} = \mathbf{\Phi}\vec{\mathbf{x}} = \mathbf{\Phi}\mathbf{\Psi}\vec{\mathbf{s}}$ 。令 $\mathbf{A} = \mathbf{\Phi}\mathbf{\Psi} \in \mathbb{R}^{m\times n}$ ，则 $\vec{\mathbf{y}} = \mathbf{A}\vec{\mathbf{s}}$ 。
4. 如果能够从 $\vec{\mathbf{y}}$ 中恢复 $\vec{\mathbf{s}}$，则能够通过 $\vec{\mathbf{x}} = \mathbf{\Psi}\vec{\mathbf{s}}$ 从 $\vec{\mathbf{y}}$ 中恢复出 $\vec{\mathbf{x}}$。

   从数学意义上来看，这种做法没有解决任何问题。因为根据 $\vec{\mathbf{y}} = \mathbf{A}\vec{\mathbf{s}}$， 从 $\vec{\mathbf{y}}$ 中恢复 $\vec{\mathbf{s}}$ 这个问题仍然是欠定的。

   但是在实际应用中发现，如果 $\vec{\mathbf{s}}$ 具有稀疏性（即大量的分量为零），则该问题能够很好地求解。这是因为稀疏性使得未知因素的影响大大减少。

   此时 $\mathbf{\Psi}$ 称作稀疏基，而 $\mathbf{A}$ 的作用类似于字典，能够将信号转换为稀疏表示。

1.5.1 压缩感知

1. 与特征选择、稀疏表示不同，压缩感知侧重的是如何利用信号本身所具有的稀疏性，从部分观测样本中恢复原信号。
2. 压缩感知分为感知测量和重构恢复两个阶段。
   - 感知测量：关注如何对原始信号进行处理以获得稀疏样本表示。常用的手段是傅里叶变换、小波变换、字典学习、稀疏编码等
   - 重构恢复：关注的是如何基于稀疏性从少量观测中恢复原信号。
3. 限定等距性`Restricted Isometry Property:RIP`：对于大小为 $m \times n, m \ll n$ 的矩阵 $\mathbf{A}$ ，若存在常数 $\delta_k \in (0,1)$ ，使得对于任意向量 $\vec{\mathbf{s}}$ 和 $\mathbf{A}$ 的所有子矩阵 $\mathbf{A}_k \in \mathbb{R}^{m\times k}$ ，都有：

$$(1-\delta_k)||\vec{\mathbf{s}}||_2^2 \le ||\mathbf{A}_k\vec{\mathbf{s}}||_2^2 \le (1+\delta_k)||\vec{\mathbf{s}}||_2^2$$

   则称 $\mathbf{A}$ 满足 k 限定等距性 `k-RIP`。

   此时通过下面的最优化问题可以近乎完美的从 $\vec{\mathbf{y}}$ 中恢复出稀疏信号 $\vec{\mathbf{s}}$ ，进而恢复出 $\vec{\mathbf{x}}$：

$$\min_{\vec{\mathbf{s}}} ||\vec{\mathbf{s}}||_0$$
$$s.t. \quad \vec{\mathbf{y}} = \mathbf{A}\vec{\mathbf{s}}$$

   这里 L_0 范数表示向量中非零元素的个数。
4. 该最优化问题涉及 L_0 范数最小化，这是个 `NP` 难问题。但是 L_1 范数最小化在一定条件下与 L_0 范数最小化问题共解，于是实际上只需要求解最小化问题：

$$\min_{\vec{\mathbf{s}}} ||\vec{\mathbf{s}}||_1$$
$$s.t. \quad \vec{\mathbf{y}} = \mathbf{A}\vec{\mathbf{s}}$$

可以将该问题转化为`LASSO`等价形式，然后通过近端梯度下降法来求解。

1.5.2 矩阵补全

1. 矩阵补全`matrix completion`解决的问题是：

$$\min_{\mathbf{X}} rank(\mathbf{X})$$
$$s.t. \quad x_{i,j} = a_{i,j}, (i,j) \in \Omega$$

其中

$$\mathbf{X} = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,n} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m,1} & x_{m,2} & \cdots & x_{m,n} \end{bmatrix} \quad \mathbf{A} = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix}$$

   - $\mathbf{A}$ 为观测矩阵，其中有很多缺失值。
   - Ω 为 $\mathbf{A}$ 中所有的有数值的下标的集合。
   - $\mathbf{X}$ 为需要恢复的稀疏信号，$rank(\mathbf{X})$ 为矩阵 $\mathbf{X}$ 的秩。

   该最优化问题也是一个`NP`难问题。

2. 考虑到 $rank(\mathbf{X})$ 在集合 $\{\mathbf{X} \in \mathbb{R}^{m\times n} : ||\mathbf{X}||_F^2 \le 1\}$ 上的凸包是 $\mathbf{X}$ 的核范数`nuclear norm`：

$$||\mathbf{X}||_* = \sum_{j=1}^{\min\{m,n\}} \sigma_j(\mathbf{X})$$

   其中 $\sigma_j(\mathbf{X})$ 表示 $\mathbf{X}$ 的奇异值。于是可以通过最小化矩阵核范数来近似求解：

$$\min_{\mathbf{X}} ||\mathbf{X}||_*$$
$$s.t. \quad x_{i,j} = a_{i,j}, (i,j) \in \Omega$$

   该最优化问题是一个凸优化问题，可以通过半正定规划`Semi-Definite Programming:SDP`求解。

3. 理论研究表明：若 $\mathbf{A}$ 的秩为 $r, n \le m$，则只需要观察 $O(mr\log^2 m)$ 个元素就能够完美恢复出 $\mathbf{A}$ 。

二、特征编码

2.1. 特征二元化

1. 特征二元化的过程是将数值型的属性转换为布尔值的属性。通常用于假设属性取值为取值分布为伯努利分布的情形。
2. 特征二元化的算法比较简单。 对属性 j 指定一个阈值 ϵ 。
   - 如果样本在属性 j 上的值大于等于 ϵ，则二元化之后为 1 。
   - 如果样本在属性 j 上的值小于 ϵ，则二元化之后为 0 。
3. 阈值 ϵ 是一个超参数，其选取需要结合模型和具体的任务来选择。

2.2. one-hot

1. 对于非数值属性，如`性别：[男，女]`、`国籍：[中国，美国，英国]`等等，可以构建一个到整数的映射。如`性别：[男，女]`属性中，将`男`映射为整数 1、`女`映射为整数 0。

   该方法的优点是简单。但是问题是，在这种处理方式中无序的属性被看成有序的。`男`和`女`无法比较大小，但是`1`和`0`有大小。

   解决的办法是采用独热码编码`One-Hot Encoding`。

2. `One-Hot Encoding`采用`N`位状态位来对`N`个可能的取值进行编码，每个取值都由独立的状态位来表示，并且在任意时刻只有其中的一位有效。

   假设属性 j 的取值为非数值的离散集合 $\{a_1, a_2, \cdots, a_K\}$ ，独热码将其扩展成 K 个属性，每个新属性代表属性 j 的一个状态位：若样本在属性 j 上的取值为 a_k，则样本在新的属性 j_k 上的取值为 1，在新的属性 $j_1, \cdots, j_{k-1}, j_{k+1}, \cdots, j_K$ 上的取值为 0 。

   - 这种做法中，如果在 $j_1, \cdots, j_K$ 上取值全为 0，则表示发生了缺失。

     > 缺失值也可以用第 $K+1$ 个状态位来表示。

   - 也可以扩展成 $K-1$ 个属性， 如果在 $j_1, \cdots, j_{K-1}$ 上取值全为 0，则表示样本在属性 j 上的取值为 a_K 。

3. `One-Hot Encoding` 的优点：

   - 能够处理非数值属性。
   - 在一定程度上也扩充了特征。如`性别`是一个属性，经过独热码编码之后变成了`是否男` 和 `是否女` 两个属性。
   - 编码后的属性是稀疏的，存在大量的零元分量。

4. 在决策树模型中，并不推荐对离散特征进行`one-hot`。 主要有两个原因：

   - 产生样本切分不平衡的问题，此时且分增益会非常小。

     如：`国籍`这个离散特征经过独热码编码之后，会产生`是否中国、是否美国、是否英国、...`等一系列特征。在这一系列特征上，只有少量样本为`1`，大量样本为`0`。

     这种划分的增益非常小，因为拆分之后：

     - 较小的那个拆分样本集，它占总样本的比例太小。无论增益多大，乘以该比例之后几乎可以忽略。
     - 较大的那个拆分样本集，它几乎就是原始的样本集，增益几乎为零。

   - 影响决策树的学习。

     决策树依赖的是数据的统计信息。而独热码编码会把数据切分到零散的小空间上。在这些零散的小空间上，统计信息是不准确的，学习效果变差。

     本质是因为独热码编码之后的特征的表达能力较差的。该特征的预测能力被人为的拆分成多份，每一份与其他特征竞争最优划分点都失败。最终该特征得到的重要性会比实际值低。

2.3 离散化

1. 离散化用于将连续的数值属性转化为离散的数值属性。

2. 是否使用特征离散化，这背后是：使用“海量离散特征+简单模型”，还是“少量连续特征+复杂模型”。

   - 对于线性模型，通常使用“海量离散特征+简单模型”。

     - 优点：模型简单。
     - 缺点：特征工程比较困难。但是一旦有成功的经验就可以推广，并且可以很多人并行研究。

   - 对于非线性模型（如深度学习），通常使用“少量连续特征+复杂模型”。

     - 优点是：不需要进行复杂的特征工程。

- 缺点是：模型复杂。

2.3.1 分桶

1. 离散化的常用方法是分桶。
   - 将所有样本在连续的数值属性 j 的取值从小到大排列 $\{a_0, a_1, \cdots, a_N\}$。
   - 然后从小到大依次选择分桶边界 $b_1, b_2, \cdots, b_M$ 。其中：
     - M 为分桶的数量，它是一个超参数，需要人工指定。
     - 每个桶的大小 $b_{k+1} - b_k$ 也是一个超参数，需要人工指定。
   - 给定属性 j 的取值 a_i ，对其进行分桶：
     - 如果 $a_i < b_1$ ，则分桶编号为 0。分桶后的属性的取值为 0 。
     - 如果 $b_k \le a_i < b_{k+1}$，则分桶编号为 k 。分桶后的属性的取值为 k 。
     - 如果 $a_i \ge b_M$ ，则分桶编号为 M 。分桶后的属性的取值为 M 。
2. 分桶的数量和边界通常需要人工指定。一般有两种方法：
   - 根据业务领域的经验来指定。如：对年收入进行分桶时，根据2017年全国居民人均可支配收入约为 2.6 万元，可以选择桶的数量为5。其中：
     - 年收入小于 1.3 万元（人均的0.5倍），则为分桶 0 。
     - 年收入在 1.3万元 ～5.2 万元（人均的0.5～2倍），则为分桶 1 。
     - 年收入在 5.3万元～26万元（人均的2倍～10倍），则为分桶 2 。
     - 年收入在 26万元～260万元（人均的10倍～100倍），则为分桶 3 。
     - 年收入超过 260万，则为分桶 4 。
   - 根据模型指定。根据具体任务来训练分桶之后的数据集，通过超参数搜索来确定最优的分桶数量和分桶边界。
3. 选择分桶大小时，有一些经验指导：
   - 分桶大小必须足够小，使得桶内的属性取值变化对样本标记的影响基本在一个不大的范围。

     即不能出现这样的情况：单个分桶的内部，样本标记输出变化很大。
   - 分桶大小必须足够大，使每个桶内都有足够的样本。

     如果桶内样本太少，则随机性太大，不具有统计意义上的说服力。
   - 每个桶内的样本尽量分布均匀。

2.3.2 特性

1. 在工业界很少直接将连续值作为逻辑回归模型的特征输入，而是将连续特征离散化为一系列 0/1 的离散特征。

   其优势有：
   - 离散化之后得到的稀疏向量，内积乘法运算速度更快，计算结果方便存储。
   - 离散化之后的特征对于异常数据具有很强的鲁棒性。

     如：销售额作为特征，当销售额在 `[30,100)` 之间时，为1，否则为 0。如果未离散化，则一个异常值 10000 会给模型造成很大的干扰。由于其数值较大，它对权重的学习影响较大。
   - 逻辑回归属于广义线性模型，表达能力受限，只能描述线性关系。特征离散化之后，相当于引入了非线性，提升模型的表达能力，增强拟合能力。

     假设某个连续特征 j ，它离散化为 M 个 0/1 特征 $j_1, j_2, \cdots, j_M$ 。则：

     $w_j * x_j \rightarrow w_{j_1} * x'_{j_1} + w_{j_2} * x'_{j_2} + \cdots + w_{j_M} * x'_{j_M}$ 。其中 $x'_{j_1}, \cdots, x'_{j_M}$ 是离散化之后的新的特征，它们的取值空间都是 $\{0, 1\}$ 。

上式右侧是一个分段线性映射，其表达能力更强。

- 离散化之后可以进行特征交叉。假设有连续特征 j，离散化为 N 个 0/1 特征；连续特征 k，离散化为 M 特 0/1 特征，则分别进行离散化之后引入了 $M+N$ 个特征。

  假设离散化时，并不是独立进行离散化，而是特征 j,k 联合进行离散化，则可以得到 $M\times N$ 个组合特征。这会进一步引入非线性，提高模型表达能力。

- 离散化之后，模型会更稳定。

  如对销售额进行离散化，`[30,100)` 作为一个区间。当销售额在40左右浮动时，并不会影响它离散化后的特征的值。

  但是处于区间连接处的值要小心处理，另外如何划分区间也是需要仔细处理。

2. 特征离散化简化了逻辑回归模型，同时降低模型过拟合的风险。

   能够对抗过拟合的原因：经过特征离散化之后，模型不再拟合特征的具体值，而是拟合特征的某个概念。因此能够对抗数据的扰动，更具有鲁棒性。

   另外它使得模型要拟合的值大幅度降低，也降低了模型的复杂度。

三、数据标准化、正则化

3.1. 数据标准化

1. 数据标准化是将样本的属性取值缩放到某个指定的范围。
2. 数据标准化的两个原因：
   - 某些算法要求样本数据的属性取值具有零均值和单位方差。
   - 样本不同属性具有不同量级时，消除数量级的影响。如下图所示为两个属性的目标函数的等高线。
     - 数量级的差异将导致量级较大的属性占据主导地位。

       从图中看到：如果样本的某个属性的量级特别巨大，将原本为椭圆的等高线压缩成直线，从而使得目标函数值仅依赖于该属性。

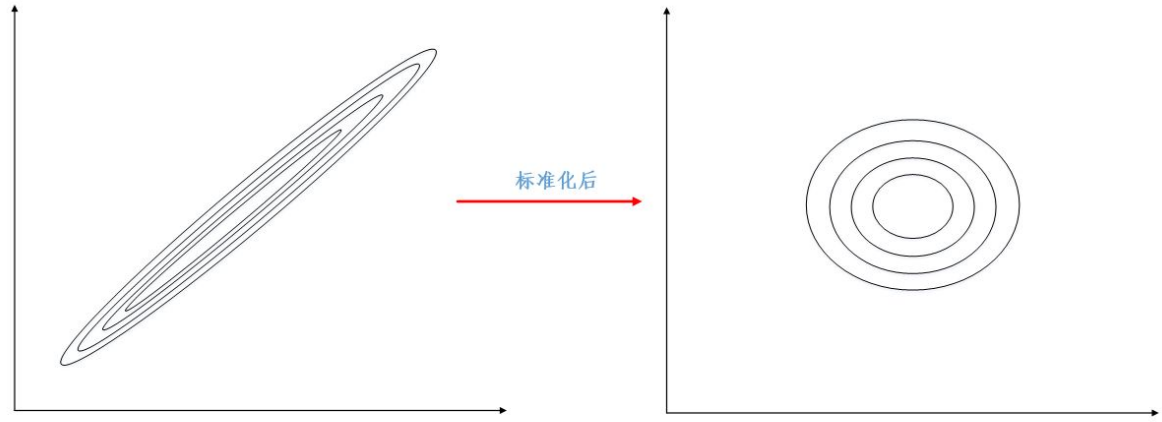
     - 数量级的差异将导致迭代收敛速度减慢。

       原始的特征进行梯度下降时，每一步梯度的方向会偏离最小值（等高线中心点）的方向，迭代次数较多，且学习率必须非常小，否则非常容易引起宽幅震荡。

     标准化后进行梯度下降时，每一步梯度的方向都几乎指向最小值（等高线中心点）的方向，迭代次数较少。

     - 所有依赖于样本距离的算法对于数据的数量级都非常敏感。

       如 k 近邻算法需要计算距离当前样本最近的 k 个样本。当属性的量级不同时，选取的最近的 k 个样本也会不同。



3. 设数据集 $D=\{(\vec{\mathbf{x}}_1,\tilde{y}_1),(\vec{\mathbf{x}}_2,\tilde{y}_2),\cdots,(\vec{\mathbf{x}}_N,\tilde{y}_N)\},\vec{\mathbf{x}}_i=(x_{i,1},\cdots,x_{i,n})^T$ 。常用的标准化算法有：
   - `min-max` 标准化：对于属性 j，设所有样本在属性 j 上的最大值为 $j_{\max}$ ，最小值为 $j_{\min}$ 。则标准化后的属性值为：

     $$\hat{x}_{i,j}=\frac{x_{i,j}-j_{\min}}{j_{\max}-j_{\min}}$$

     标准化之后，所有样本在属性 j 上的取值都在 `[0,1]` 之间。
   - `z-score` 标准化：对于属性 j，设所有样本在属性 j 上的均值为 μ_j ，方差为 σ_j 。则标准化后的属性值为：

     $$\hat{x}_{i,j}=\frac{x_{i,j}-\mu_j}{\sigma_j}$$

     标准化之后，样本集的所有属性的均值都为 0，标准差均为 1。
4. 注意：如果数据集分为训练集、验证集和测试集，则：训练集、验证集、测试集使用相同标准化参数，该参数的值都是从训练集中得到。
   - 如果使用 `min-max` 标准化，则属性 j 的标准化参数 $j_{\max},j_{\min}$ 都是从训练集中计算得到。
   - 如果使用 `z-score` 标准化，则属性 j 的标准化参数 μ_j,σ_j 都是从训练集中计算得到。

3.2. 数据正则化

1. 数据正则化是将样本的某个范数（如 L_1 范数）缩放到单位1。

   设数据集 $D=\{(\vec{\mathbf{x}}_1,\tilde{y}_1),(\vec{\mathbf{x}}_2,\tilde{y}_2),\cdots,(\vec{\mathbf{x}}_N,\tilde{y}_N)\},\vec{\mathbf{x}}_i=(x_{i,1},x_{i,2},\cdots,x_{i,n})^T$。 则样本 $\vec{\mathbf{x}}_i$ 正则化后的结果为：

   $$\hat{\vec{\mathbf{x}}}_i=\left(\frac{x_{i,1}}{L_p(\vec{\mathbf{x}}_i)},\frac{x_{i,2}}{L_p(\vec{\mathbf{x}}_i)},\cdots,\frac{x_{i,n}}{L_p(\vec{\mathbf{x}}_i)}\right)^T$$

   其中 L_p 为范数：$L_p(\vec{\mathbf{x}}_i)=(|x_{i,1}|^p+|x_{i,2}|^p+\cdots+|x_{i,n}|^p)^{1/p}$ 。
2. 正则化的过程是针对单个样本的，对每个样本将它缩放到单位范数。

   标准化是针对单个属性的，需要用到所有样本在该属性上的值。
3. 通常如果使用二次型（如点积）或者其他核方法计算两个样本之间的相似性时，该方法会很有用。

四、特征选择

1. 对于一个学习任务，给定了属性集，其中某些属性可能对于学习来说是很关键的，但是有些属性可能就意义不大。
   - 对当前学习任务有用的属性称作相关特征`relevant feature`。
   - 对当前学习任务没有用的属性称作无关特征`irrelevant feature`。

   从给定的特征集合中选出相关特征子集的过程称作特征选择`feature selection`。
2. 特征选择可能会降低模型的预测能力。因为被剔除的特征中可能包含了有效的信息，抛弃了这部分信息会一定程度上降低预测准确率。

   这是计算复杂度和预测能力之间的折衷：
   - 如果保留尽可能多的特征，则模型的预测能力会有所提升，但是计算复杂度会上升。
   - 如果剔除尽可能多的特征，则模型的预测能力会有所下降，但是计算复杂度会下降。

4.1 特征选择原理

1. 特征选择是一个重要的数据预处理（`data preprocessing`）过程。在现实机器学习任务中，获取数据之后通常首先进行特征选择，然后再训练学习器。

   进行特征选择的原因：
   - 首先，在现实任务中经常会遇到维数灾难问题，这是由于属性过多造成的。如果能从中选择出重要的特征，使得后续学习过程仅仅需要在一部分特征上构建模型，则维数灾难问题会大大减轻。

     从这个意义上讲，特征选择与降维技术有相似的动机。事实上它们是处理高维数据的两大主流技术。
   - 其次，去除不相关特征往往会降低学习任务的难度。
2. 特征选择过程必须确保不丢失重要特征，否则后续学习过程会因为重要信息的缺失而无法获得很好的性能。
   - 给定数据集，如果学习任务不同，则相关特征很可能不同，因此特征选择中的无关特征指的是与当前学习任务无关的特征。
   - 有一类特征称作冗余特征`redundant feature`，它们所包含的信息能从其他特征中推演出来。
     - 冗余特征在很多时候不起作用，去除它们能够减轻学习过程的负担。
     - 但如果冗余特征恰好对应了完成学习任务所需要的某个中间概念，则该冗余特征是有益的，能降低学习任务的难度。

     这里暂且不讨论冗余特征，且假设初始的特征集合包含了所有的重要信息。
3. 要想从初始的特征集合中选取一个包含了所有重要信息的特征子集，如果没有任何领域知识作为先验假设，则只能遍历所有可能的特征组合。

   这在计算上是不可行的，因为这样会遭遇组合爆炸，特征数量稍多就无法进行。

   一个可选的方案是：
   - 产生一个候选子集，评价出它的好坏。
   - 基于评价结果产生下一个候选子集，再评价其好坏。
   - 这个过程持续进行下去，直至无法找到更好的后续子集为止。

   这里有两个问题：如何根据评价结果获取下一个候选特征子集？如何评价候选特征子集的好坏？

4.1.1 子集搜索

1. 如何根据评价结果获取下一个候选特征子集？这是一个子集搜索`subset search`问题。

2. 解决该问题的算法步骤如下：
   - 给定特征集合 $\mathbb{A}=\{A_1,A_2,\cdots,A_d\}$ ，首先将每个特征看作一个候选子集（即每个子集中只有一个元素），然后对这 d 个候选子集进行评价。

     假设 A_2 最优，于是将 A_2 作为第一轮的选定子集。
   - 然后在上一轮的选定子集中加入一个特征，构成了包含两个特征的候选子集。

     假定 A_2,A_5 最优，且优于 A_2 ，于是将 A_2,A_5 作为第二轮的选定子集。
   - ....
   - 假定在第 $k+1$ 轮时，本轮的最优的特征子集不如上一轮的最优的特征子集，则停止生成候选子集，并将上一轮选定的特征子集作为特征选择的结果。
3. 这种逐渐增加相关特征的策略称作前向`forward`搜索。

   类似地，如果从完整的特征集合开始，每次尝试去掉一个无关特征，这样逐渐减小特征的策略称作后向`backward`搜索。
4. 也可以将前向和后向搜索结合起来，每一轮逐渐增加选定相关特征（这些特征在后续迭代中确定不会被去除）、同时减少无关特征，这样的策略被称作双向`bidirectional`搜索。
5. 该策略是贪心的，因为它们仅仅考虑了使本轮选定集最优。但是除非进行穷举搜索，否则这样的问题无法避免。

4.1.2 子集评价

1. 如何评价候选特征子集的好坏？这是一个子集评价`subset evaluation`问题。
2. 给定数据集 $\mathbb{D}$，假设所有属性均为离散型。对属性子集 $\mathbb{A}$ ， 假定根据其取值将 $\mathbb{D}$ 分成了 V 个子集：$\{\mathbb{D}_1,\mathbb{D}_2,\cdots,\mathbb{D}_V\}$

   于是可以计算属性子集 $\mathbb{A}$ 的信息增益：

   $$g(\mathbb{D},\mathbb{A})=H(\mathbb{D})-H(\mathbb{D}\mid\mathbb{A})=H(\mathbb{D})-\sum_{v=1}^{V}\frac{|\mathbb{D}_v|}{|\mathbb{D}|}H(\mathbb{D}_v)$$

   其中 $|\cdot|$ 为集合大小， $H(\cdot)$ 为熵。

   信息增益越大，则表明特征子集 $\mathbb{A}$ 包含的有助于分类的信息越多。于是对于每个候选特征子集，可以基于训练数据集 $\mathbb{D}$ 来计算其信息增益作为评价准则。
3. 更一般地，特征子集 $\mathbb{A}$ 实际上确定了对数据集 $\mathbb{D}$ 的一个划分规则。
   - 每个划分区域对应着 $\mathbb{A}$ 上的一个取值，而样本标记信息 y 则对应着 $\mathbb{D}$ 的真实划分。
   - 通过估算这两种划分之间的差异，就能对 $\mathbb{A}$ 进行评价：与 y 对应的划分的差异越小，则说明 $\mathbb{A}$ 越好。
   - 信息熵仅仅是判断这个差异的一种方法，其他能判断这两个划分差异的机制都能够用于特征子集的评价。
4. 将特征子集搜索机制与子集评价机制结合就能得到特征选择方法。
   - 事实上，决策树可以用于特征选择，所有树结点的划分属性所组成的集合就是选择出来的特征子集。
   - 其他特征选择方法本质上都是显式或者隐式地结合了某些子集搜索机制和子集评价机制。
5. 常见的特征选择方法大致可分为三类：过滤式`filter`、包裹式`wrapper`、嵌入式`embedding` 。

4.2 过滤式选择

1. 过滤式方法先对数据集进行特征选择，然后再训练学习器，特征选择过程与后续学习器无关。

   这相当于先用特征选择过程对初始特征进行过滤，再用过滤后的特征来训练模型。

2. `Relief:Relevant Features`是一种著名的过滤式特征选择方法，该方法设计了一个相关统计量来度量特征的重要性。

   - 该统计量是一个向量，其中每个分量都对应于一个初始特征。特征子集的重要性则是由该子集中每个特征所对应的相关统计量分量之和来决定的。
   - 最终只需要指定一个阈值 τ ，然后选择比 τ 大的相关统计量分量所对应的特征即可。

     也可以指定特征个数 k ，然后选择相关统计量分量最大的 k 个特征。

3. 给定训练集 $\mathbb{D}=\{(\vec{\mathbf{x}}_1,\tilde{y}_1),(\vec{\mathbf{x}}_2,\tilde{y}_2),\cdots,(\vec{\mathbf{x}}_N,\tilde{y}_N)\},\tilde{y}_i\in\{0,1\}$ 。 对于每个样本 $\vec{\mathbf{x}}_i$ ：

   - `Relief` 先在 $\vec{\mathbf{x}}_i$ 同类样本中寻找其最近邻 $\vec{\mathbf{x}}_{nh_i}$ ，称作猜中近邻`near-hit` 。
   - 然后从 $\vec{\mathbf{x}}_i$ 的异类样本中寻找其最近邻 $\vec{\mathbf{x}}_{nm_i}$ ，称作猜错近邻`near-miss` 。
   - 然后相关统计量对应于属性 j 的分量为：

     $$\delta_j=\sum_{i=1}^{N}\left(-\text{diff}(x_{i,j},x_{nh_i,j})^2+\text{diff}(x_{i,j},x_{nm_i,j})^2\right)$$

     其中 $\text{diff}(x_{a,j},x_{b,j})$ 为两个样本在属性 j 上的差异值，其结果取决于该属性是离散的还是连续的：

     - 如果属性 j 是离散的，则：

       $$\text{diff}(x_{a,j},x_{b,j})=\begin{cases}0, & \text{if} \quad x_{a,j}=x_{b,j}\\ 1, & else\end{cases}$$

     - 如果属性 j 是连续的，则：

       $$\text{diff}(x_{a,j},x_{b,j})=|x_{a,j}-x_{b,j}|$$

       注意：此时 $x_{a,j},x_{b,j}$ 需要标准化到 `[0,1]` 区间。

4. 从公式

   $$\delta_j=\sum_{i=1}^{N}\left(-\text{diff}(x_{i,j},x_{nh_i,j})^2+\text{diff}(x_{i,j},x_{nm_i,j})^2\right)$$

   可以看出：

   - 如果 $\vec{\mathbf{x}}_i$ 与其猜中近邻 $\vec{\mathbf{x}}_{nh_i}$ 在属性 j 上的距离小于 $\vec{\mathbf{x}}_i$ 与其猜错近邻 $\vec{\mathbf{x}}_{nm_i}$ 的距离，则说明属性 j 对于区分同类与异类样本是有益的，于是增大属性 j 所对应的统计量分量。
   - 如果 $\vec{\mathbf{x}}_i$ 与其猜中近邻 $\vec{\mathbf{x}}_{nh_i}$ 在属性 j 上的距离大于 $\vec{\mathbf{x}}_i$ 与其猜错近邻 $\vec{\mathbf{x}}_{nm_i}$ 的距离，则说明属性 j 对于区分同类与异类样本是起负作用的，于是减小属性 j 所对应的统计量分量。
   - 最后对基于不同样本得到的估计结果进行平均，就得到各属性的相关统计量分量。分量值越大，则对应属性的分类能力越强。

5. `Relief` 是为二分类问题设计的，其扩展变体 `Relief-F` 能处理多分类问题。

   假定数据集 $\mathbb{D}$ 中的样本类别为： $c_1,c_2,\cdots,c_K$ 。对于样本 $\vec{\mathbf{x}}_i$ ，假设 $\tilde{y}_i=c_k$ 。

   - `Relief-F` 先在类别 c_k 的样本中寻找 $\vec{\mathbf{x}}_i$ 的最近邻 $\vec{\mathbf{x}}_{nh_i}$ 作为猜中近邻。
   - 然后在 c_k 之外的每个类别中分别找到一个 $\vec{\mathbf{x}}_i$ 的最近邻 $\vec{\mathbf{x}}_{nm_i^l},l=1,2,\cdots,K;l\neq k$ 作为猜错近邻。
   - 于是相关统计量对应于属性 j 的分量为：

     $$\delta_j=\sum_{i=1}^{N}\left(-\text{diff}(x_{i,j},x_{nh_i,j})^2+\sum_{l\neq k}\left(p_l\times\text{diff}(x_{i,j},x_{nm_i^l,j})^2\right)\right)$$

     其中 p_l 为第 l 类的样本在数据集 $\mathbb{D}$ 中所占的比例。

4.3 包裹式选择

1. 与过滤式特征选择不考虑后续学习器不同，包裹式特征选择直接把最终将要使用的学习器的性能作为特征子集的评价准则。其目的就是为给定学习器选择最有利于其性能、量身定做的特征子集。
   - 优点：由于直接针对特定学习器进行优化，因此从最终学习器性能来看，效果比过滤式特征选择更好。
   - 缺点：需要多次训练学习器，因此计算开销通常比过滤式特征选择大得多。
2. `LVW:Las Vegas Wrapper`是一个典型的包裹式特征选择方法。它是`Las Vegas method`框架下使用随机策略来进行子集搜索，并以最终分类器的误差作为特征子集的评价标准。
3. `LVW`算法：
   - 输入：
     - 数据集 $\mathbb{D}=\{(\vec{\mathbf{x}}_1,\tilde{y}_1),(\vec{\mathbf{x}}_2,\tilde{y}_2),\cdots,(\vec{\mathbf{x}}_N,\tilde{y}_N)\}$
     - 特征集 $\mathbb{A}=\{1,2,\cdots,n\}$
     - 学习器 `estimator`
     - 迭代停止条件 T
   - 输出： 最优特征子集 $\mathbb{A}^*$
   - 算法步骤：
     - 初始化：令候选的最优特征子集 $\tilde{\mathbb{A}}^*=\mathbb{A}$，然后学习器`estimator`在特征子集 $\tilde{\mathbb{A}}^*$ 上使用交叉验证法进行学习，通过学习结果评估学习器`estimator`的误差 err^*。
     - 迭代，停止条件为迭代次数到达 T 。迭代过程为：
       - 随机产生特征子集 $\mathbb{A}'$ 。
       - 学习器`estimator`在特征子集 $\mathbb{A}'$ 上使用交叉验证法进行学习，通过学习结果评估学习器`estimator`的误差 err' 。
       - 如果 err' 比 err^* 更小，或者 $err'=err^*$ 但是 $\mathbb{A}'$ 的特征数量比 $\tilde{\mathbb{A}}^*$ 的特征数量更少，则将 $\mathbb{A}'$ 作为候选的最优特征子集：$\tilde{\mathbb{A}}^*=\mathbb{A}';\quad err^*=err'$ 。
     - 最终 $\mathbb{A}^*=\tilde{\mathbb{A}}^*$。
4. 由于`LVW`算法中每次特征子集评价都需要训练学习器，计算开销很大，因此算法设置了停止条件控制参数 T 。

   但是如果初始特征数量很多、T 设置较大、以及每一轮训练的时间较长， 则很可能算法运行很长时间都不会停止。即：如果有运行时间限制，则有可能给不出解。

4.4 嵌入式选择

1. 在过滤式和包裹式特征选择方法中，特征选择过程与学习器训练过程有明显的分别。

   嵌入式特征选择是将特征选择与学习器训练过程融为一体，两者在同一个优化过程中完成的。即学习器训练过程中自动进行了特征选择。

2. 以线性回归模型为例。

   给定数据集 $\mathbb{D}=\{(\vec{\mathbf{x}}_1,\tilde{y}_1),(\vec{\mathbf{x}}_2,\tilde{y}_2),\cdots,(\vec{\mathbf{x}}_N,\tilde{y}_N)\},\tilde{y}_i\in\mathbb{R}$ 。 以平方误差为损失函数，则优化目标为：

   $$\min_{\vec{\mathbf{w}}}\sum_{i=1}^{N}(\tilde{y}_i-\vec{\mathbf{w}}^T\vec{\mathbf{x}}_i)^2$$

   - 如果使用 L_2 范数正则化，则优化目标为：

   $$\min_{\vec{\mathbf{w}}}\sum_{i=1}^{N}(\tilde{y}_i-\vec{\mathbf{w}}^T\vec{\mathbf{x}}_i)^2+\lambda||\vec{\mathbf{w}}||_2^2,\quad \lambda>0$$

此时称作岭回归`ridge regression`。

- 如果使用 L_1 范数正则化，则优化目标为：

$$\min_{\vec{\mathbf{w}}} \sum_{i=1}^{N}(\tilde{y}_i - \vec{\mathbf{w}}^T\vec{\mathbf{x}}_i)^2 + \lambda||\vec{\mathbf{w}}||_1, \quad \lambda > 0$$

此时称作`LASSO:Least Absolute Shrinkage and Selection Operator`回归。

3. 引入 L_1 范数除了降低过拟合风险之外，还有一个好处：它求得的 $\vec{\mathbf{w}}$ 会有较多的分量为零。即：它更容易获得稀疏解。

   于是基于 L_1 正则化的学习方法就是一种嵌入式特征选择方法，其特征选择过程与学习器训练过程融为一体，二者同时完成。

4. L_1 正则化问题的求解可以用近端梯度下降`Proximal Gradient Descent:PGD`算法求解。

   对于优化目标：$\min_{\vec{\mathbf{x}}} f(\vec{\mathbf{x}}) + \lambda||\vec{\mathbf{x}}||_1$ ，若 $f(\vec{\mathbf{x}})$ 可导且 ∇f 满足`L-Lipschitz`条件，即存在常数 $L > 0$ 使得：

$$||\nabla f(\vec{\mathbf{x}}) - \nabla f(\vec{\mathbf{x}}')||_2^2 \le L||\vec{\mathbf{x}} - \vec{\mathbf{x}}'||_2^2, \ \forall(\vec{\mathbf{x}}, \vec{\mathbf{x}}')$$

   则在 $\vec{\mathbf{x}}_0$ 附近将 $f(\vec{\mathbf{x}})$ 通过二阶泰勒公式展开的近似值为：

$$\begin{aligned}\hat{f}(\vec{\mathbf{x}}) &\simeq f(\vec{\mathbf{x}}_0) + \nabla f(\vec{\mathbf{x}}_0)\cdot(\vec{\mathbf{x}} - \vec{\mathbf{x}}_0) + \frac{L}{2}||\vec{\mathbf{x}} - \vec{\mathbf{x}}_0||_2^2\\ &= \frac{L}{2}||\vec{\mathbf{x}} - (\vec{\mathbf{x}}_0 - \frac{1}{L}\nabla f(\vec{\mathbf{x}}_0))||_2^2 + const\end{aligned}$$

   其中 $const$ 是与 $\vec{\mathbf{x}}$ 无关的常数项。

   - 若通过梯度下降法对 $f(\vec{\mathbf{x}})$ 进行最小化，则每一步梯度下降迭代实际上等价于最小化二次函数 $\hat{f}(\vec{\mathbf{x}})$。
   - 同理，若通过梯度下降法对 $f(\vec{\mathbf{x}}) + \lambda||\vec{\mathbf{x}}||_1$ 进行最小化，则每一步梯度下降迭代实际上等价于最小化函数：$\hat{f}(\vec{\mathbf{x}}) + \lambda||\vec{\mathbf{x}}||_1$。

     则每一步迭代为：

$$\vec{\mathbf{x}}^{<k+1>} = \arg\min_{\vec{\mathbf{x}}} \frac{L}{2}||\vec{\mathbf{x}} - (\vec{\mathbf{x}}^{<k>} - \frac{1}{L}\nabla f(\vec{\mathbf{x}}^{<k>}))||_2^2 + \lambda||\vec{\mathbf{x}}||_1$$

     其中 $\vec{\mathbf{x}}^{<k>}$ 为 $\vec{\mathbf{x}}$ 的第 k 次迭代的值。

     该问题有解析解，因此通过`PGD`能够使得`LASSO`和其他基于 L_1 范数最小化的方法能够快速求解。

5. 常见的嵌入式选择模型：
   - 在`Lasso`中，λ 参数控制了稀疏性：
     - 如果 λ 越小，则稀疏性越小，则被选择的特征越多。
     - 如果 λ 越大，则稀疏性越大，则被选择的特征越少。
   - 在`SVM`和`logistic-regression`中，参数`C`控制了稀疏性
     - 如果`C`越小，则稀疏性越大，则被选择的特征越少。
     - 如果`C`越大，则稀疏性越小，则被选择的特征越多。

五、稀疏表示和字典学习

1. 对于 $\mathbb{D} = \{(\vec{\mathbf{x}}_1, \tilde{y}_1), (\vec{\mathbf{x}}_2, \tilde{y}_2), \cdots, (\vec{\mathbf{x}}_N, \tilde{y}_N)\}, \tilde{y}_i \in \mathbb{R}$。构

   建矩阵 $\mathbf{D} = (\vec{\mathbf{x}}_1, \vec{\mathbf{x}}_2, \cdots, \vec{\mathbf{x}}_N)^T$ ，其内容为：

$$\mathbf{D}=\begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,n} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N,1} & x_{N,2} & \cdots & x_{N,n} \end{bmatrix}$$

其中每一行对应一个样本，每一列对应一个特征。

2. 特征选择所考虑的问题是：矩阵 $\mathbf{D}$ 中的许多列与当前学习任务无关。

   通过特征选择去除这些列，则学习器训练过程仅需要在较小的矩阵上进行。这样学习任务的难度可能有所降低，涉及的计算和存储开销会减少，学得模型的可解释性也会提高。

3. 考虑另一种情况： $\mathbf{D}$ 中有大量元素为 0 ，这称作稀疏矩阵。

   当数据集具有这样的稀疏表达形式时，对学习任务来讲会有不少好处：

   - 如果数据集具有高度的稀疏性，则该问题很可能是线性可分的。对于线性支持向量机，这种情况能取得更佳的性能。
   - 稀疏样本并不会造成存储上的巨大负担，因为稀疏矩阵已经有很多很高效的存储方法。

4. 现在问题是：如果给定数据集 $\mathbb{D}$ 是稠密的（即普通的、非稀疏的），则能否将它转化为稀疏的形式？

   这就是字典学习 `dictionary learning` 和稀疏编码 `sparse coding` 的目标。

5.1 原理

1. 字典学习：学习一个字典，通过该字典将样本转化为合适的稀疏表示形式。它侧重于学得字典的过程。

   稀疏编码：获取样本的稀疏表达，不一定需要通过字典。它侧重于对样本进行稀疏表达的过程。

   这两者通常是在同一个优化求解过程中完成的，因此这里不做区分，统称为字典学习。

2. 给定数据集 $\mathbb{D}=\{(\vec{\mathbf{x}}_1,\tilde{y}_1),(\vec{\mathbf{x}}_2,\tilde{y}_2),\cdots,(\vec{\mathbf{x}}_N,\tilde{y}_N)\}$ ，希望对样本 $\vec{\mathbf{x}}_i$ 学习到它的一个稀疏表示 $\vec{\alpha}_i\in\mathbb{R}^k$ 。其中 $\vec{\alpha}_i$ 是一个 k 维列向量，且其中大量元素为 0 。

   一个自然的想法进行线性变换，即寻找一个矩阵 $\mathbf{P}\in\mathbb{R}^{k\times n}$ 使得 $\mathbf{P}\vec{\mathbf{x}}_i=\vec{\alpha}_i$ 。

3. 现在的问题是：既不知道变换矩阵 $\mathbf{P}$ ，也不知道 $\vec{\mathbf{x}}_i$ 的稀疏表示 $\vec{\alpha}_i$。

   因此求解的目标是：

   - 根据 $\vec{\alpha}_i$ 能正确还原 $\vec{\mathbf{x}}_i$，或者还原的误差最小。
   - $\vec{\alpha}_i$ 尽量稀疏，即它的分量尽量为零。

   因此给出字典学习的最优化目标：

   $$\min_{\mathbf{B},\vec{\alpha}_i}\sum_{i=1}^{N}||\vec{\mathbf{x}}_i-\mathbf{B}\vec{\alpha}_i||_2^2+\lambda\sum_{i=1}^{N}||\vec{\alpha}_i||_1$$

   其中 $\mathbf{B}\in\mathbb{R}^{n\times k}$ 称作字典矩阵。 k 称作字典的词汇量，通常由用户指定来控制字典的规模，从而影响到稀疏程度。

   - 上式中第一项希望 $\vec{\alpha}_i$ 能够很好地重构 $\vec{\mathbf{x}}_i$ 。
   - 第二项则希望 $\vec{\alpha}_i$ 尽可能稀疏。

5.2 算法

1. 求解该问题采用类似 `LASSO` 的解法，但是使用变量交替优化的策略：

   - 第一步：固定字典 $\mathbf{B}$， 为每一个样本 $\vec{\mathbf{x}}_i$ 找到相应的 $\vec{\alpha}_i$ ：

$$\min_{\vec{\alpha}_i}||\vec{\mathbf{x}}_i-\mathbf{B}\vec{\alpha}_i||_2^2+\lambda\sum_{i=1}^{N}||\vec{\alpha}_i||_1$$

- 第二步：根据下式，以 $\vec{\alpha}_i$ 为初值来更新字典 $\mathbf{B}$ ，即求解：$\min_{\mathbf{B}}||\mathbf{X}-\mathbf{B}\mathbf{A}||_F^2$ 。

  其中 $\mathbf{X}=(\vec{\mathbf{x}}_1,\vec{\mathbf{x}}_2,\cdots,\vec{\mathbf{x}}_N)\in\mathbb{R}^{n\times N}$ ， $\mathbf{A}=(\vec{\alpha}_1,\vec{\alpha}_2,\cdots,\vec{\alpha}_N)\in\mathbb{R}^{k\times N}$ 。写成矩阵的形式为：

$$\mathbf{X}=\begin{bmatrix}x_{1,1}&x_{2,1}&\cdots&x_{N,1}\\x_{1,2}&x_{2,2}&\cdots&x_{N,2}\\\vdots&\vdots&\ddots&\vdots\\x_{1,n}&x_{2,n}&\cdots&x_{N,n}\end{bmatrix}\quad\mathbf{A}=\begin{bmatrix}\alpha_{1,1}&\alpha_{2,1}&\cdots&\alpha_{N,1}\\\alpha_{1,2}&\alpha_{2,2}&\cdots&\alpha_{N,2}\\\vdots&\vdots&\ddots&\vdots\\\alpha_{1,n}&\alpha_{2,n}&\cdots&\alpha_{N,n}\end{bmatrix}$$

  这里 $||\cdot||_F$ 为矩阵的 `Frobenius` 范数（所有元素的平方和的平方根）。对于矩阵 $\mathbf{M}$ ， 有 $||\mathbf{M}||_F=\sqrt{\sum_i\sum_j|m_{ij}|^2}$

- 反复迭代上述两步，最终即可求得字典 $\mathbf{B}$ 和样本 $\vec{\mathbf{x}}_i$ 的稀疏表示 $\vec{\alpha}_i$。

2. 这里有个最优化问题：

$$\min_{\mathbf{B}}||\mathbf{X}-\mathbf{B}\mathbf{A}||_F^2$$

   该问题有多种求解方法，常用的有基于逐列更新策略的 `KSVD` 算法。

   令 $\vec{\mathbf{b}}_i$ 为字典矩阵 $\mathbf{B}$ 的第 i 列， $\vec{\mathbf{a}}^j$ 表示稀疏矩阵 $\mathbf{A}$ 的第 j 行。 固定 $\mathbf{B}$ 其他列，仅考虑第 i 列，则有：

$$\min_{\vec{\mathbf{b}}_i}||\mathbf{X}-\sum_{j=1}^{k}\vec{\mathbf{b}}_i\vec{\mathbf{a}}^j||_F^2=\min_{\vec{\mathbf{b}}_i}||(\mathbf{X}-\sum_{j=1,j\neq i}^{k}\vec{\mathbf{b}}_i\vec{\mathbf{a}}^j)-\vec{\mathbf{b}}_i\vec{\mathbf{a}}^i||_F^2$$

   令 $\mathbf{E}_i=\mathbf{X}-\sum_{j=1,j\neq i}^{k}\vec{\mathbf{b}}_i\vec{\mathbf{a}}^j$ ，它表示去掉 $\vec{\mathbf{x}}_i$ 的稀疏表示之后，样本集的稀疏表示与原样本集的误差矩阵。

   考虑到更新字典的第 i 列 $\vec{\mathbf{b}}_i$ 时，其他各列都是固定的，则 $\mathbf{E}_i$ 是固定的。则最优化问题转换为：

$$\min_{\vec{\mathbf{b}}_i}||\mathbf{E}_i-\vec{\mathbf{b}}_i\vec{\mathbf{a}}^i||_F^2$$

   求解该最优化问题只需要对 $\mathbf{E}_i$ 进行奇异值分解以取得最大奇异值所对应的正交向量。

3. 直接对 $\mathbf{E}_i$ 进行奇异值分解会同时修改 $\vec{\mathbf{b}}_i$ 和 $\vec{\mathbf{a}}^i$， 从而可能破坏 $\mathbf{A}$ 的稀疏性。因为第二步 “以 $\vec{\alpha}_i$ 为初值来更新字典 $\mathbf{B}$ ” 中， 在更新 $\mathbf{B}$ 前后 $\vec{\alpha}_i$ 的非零元所处的位置和非零元素的值很可能不一致。

   为避免发生这样的情况 `KSVD` 对 $\mathbf{E}_i$ 和 $\vec{\mathbf{a}}^i$ 进行了专门处理：

   - $\vec{\mathbf{a}}^i$ 仅保留非零元素。
   - $\mathbf{E}_i$ 仅保留 $\vec{\mathbf{b}}_i$ 和 $\vec{\mathbf{a}}^i$ 的非零元素的乘积项，然后再进行奇异值分解，这样就保持了第一步得到的稀疏性。

六、多类分类问题

1. 某些算法原生的支持多分类，如：决策树、最近邻算法等。但是有些算法只能求解二分类问题，如：支持向量机。
2. 对于只能求解二分类问题的算法，一旦遇到问题是多类别的，那么可以将多分类问题拆解成二分类任务求解。

   即：

   - 先对原问题进行拆分，然后为拆出的每个二分类任务训练一个分类器。

- 测试时，对这些二分类器的预测结果进行集成，从而获得最终的多分类结果。

3. 多分类问题有三种拆解方式：
   - 一对其余(`One-vs-rest:OvR`)。
   - 一对一(`one-vs-one:OvO`)。
   - 多对多(`many-vs-many:MvM`)。

6.1 one vs rest

1. 一对其余：为每一个类别训练一个分类器。

   假设类别为 $\{c_1,c_2,\cdots,c_K\}$ ，则训练 K 个分类器 $CLF_1,CLF_2,\cdots,CLF_K$ ：

   - 训练 CLF_i 时，将类别为 c_i 的样本点定义为正类，将类别不是 c_i 的样本点定义为负类。
   - 训练 CLF_i 不光需要给出预测结果是否属于类别 c_i，还要给出置信度。

2. 预测时，对于未知的实例，用训练出来的 K 个分类器来预测。

   假设置信度最高的分类器为 CLF_m ，则该实例的类别预测为 c_m 。

3. 缺点：非常容易陷入样本不平衡。

   即使训练集中每一类样本都是平衡的，训练每个分类器时样本反而不平衡。

6.2 one vs one

1. 一对一：为每一对类别训练一个分类器。

   假设类别为 $\{c_1,c_2,\cdots,c_K\}$。那么训练 $\frac{K(K-1)}{2}$ 个分类器 $CLF_{1,2},CLF_{1,3},\cdots,CLF_{i,j},\cdots,CLF_{K-1,K}$。

   $CLF_{i,j},i<j$ 分类器从原始训练集中提取类别为 c_i,c_j 的样本点作为新的训练集，然后训练 $CLF_{i,j}$ 。

2. 预测时，对于未知的实例，对预测结果进行投票。
   - 首先设投票结果为 $s_0=0,s_1=0,\cdots,s_K=0$
   - 然后用每个分类器 $CLF_{i,j},\ i<j;i,j=1,2,\cdots,K$ 对未知实例进行预测：
     - 若预测结果是类别 c_i，则 $s_i+=1$。
     - 若预测结果是类别 c_j，则 $s_j+=1$。
   - 最终假设 s_m 最大，则该未知的实例分类为 c_m 。

3. 缺点：需要训练的分类器数量为 $O(K^2)$ ，计算量太大。

6.3 many vs many

1. 多对多：每次都将若干个类作为正类，若干个其他类作为反类。
   - 正、反类的构造必须有特殊的设计，不能随意选取。
   - 通常采用纠错输出码 `Error Correcting Output Codes:ECOC` 技术。该技术将编码的思想引入类别拆分，并尽可能在解码过程中具有容错性。

2. `ECOC` 工作过程主要分两步，假设类别为 $c_1,c_2,\cdots,c_K$：
   - 编码：对 K 个类别进行 M 次划分，每次划分都将一部分类别划分为正类，一部分类别划分为反类，从而形成一个二分类训练集。

     这样一个产生M 个训练集，可以训练出 M个分类器。
   - 解码：用 M 个分类器分别对测试样本进行预测，这些预测标记组成一个编码。

     将这个预测编码与每个类别各自的编码进行比较，返回其中距离最小的类别作为最终预测结果。

七、类别不平衡问题

1. 通常在机器学习中都有一个基本假设：不同类别的训练样本数目相当。
   - 如果不同类别的训练样本数目稍有差别，通常影响不大。
   - 如果不同类别的训练样本数目差别很大（极端情况下，如正类样本只有十个，反类样本一百万个），则会对学习过程造成影响。这就是类别不平衡问题(`class-imbalance`)。

     这里讨论中，假设正类样本偏少、反类样本偏多。
2. 对于类别不平衡问题，常用的有三种方法：
   - 基于再缩放策略进行决策，称之为阈值移动`threshold-moving` 。
   - 直接对训练集里的反类样本进行欠采样`undersampling`。
   - 直接对训练集里的正类样本进行过采样`oversampling`。
3. 对于正负样本极不平衡的场景，可以完全换一个不同的角度来看问题：将它看作一分类`One Class Learning`或者异常检测`Novelty Detection`问题。

   此时可以用`One-class SVM`模型。

7.1 再缩放

1. 假设对样本 $\vec{\mathbf{x}}$ 进行分类时，预测为正类的概率为 p。常规的做法是将 p 与一个阈值，比如 0.5 ， 进行比较。如果 $p>0.5$ 时，就判别该样本为正类。

   概率 p 刻画了样本为正类的可能性， 几率 $\frac{p}{1-p}$ 刻画了正类可能性与反类可能性的比值。
2. 当存在类别不平衡时，假设 N^+ 表示正类样本数目，N^- 表示反类样本数目，则观测几率是 $\frac{N^+}{N^-}$。

   假设训练集是真实样本总体的无偏采样，因此可以用观测几率代替真实几率。于是只要分类器的预测几率高于观测几率就应该判断为正类。即如果 $\frac{p}{1-p}>\frac{N^+}{N^-}$ ， 则预测为正类。
3. 通常分类器都是基于概率值来进行预测的，因此需要对其预测值进行调整。在进行预测的时候，令：

   $$\frac{\bar{p}}{1-\bar{p}}=\frac{p}{1-p}\times\frac{N^-}{N^+}$$

   然后再将 $\bar{p}$ 跟阈值比较。这就是类别不平衡学习的一个基本策略：再缩放`rescalling` 。
4. 再缩放虽然简单，但是由于“训练集是真实样本总体的无偏采样”这个假设往往不成立，所以无法基于训练集观测几率来推断出真实几率。

7.2 欠采样

1. 欠采样会去除一些反类使得正、反类数目接近。
2. 欠采样若随机抛弃反类，则可能丢失一些重要信息。

   常用方法是将反类划分成若干个集合供不同学习器使用，这样对每个学习器来看都是欠采样，但是全局来看并不会丢失重要信息。

7.3 过采样

1. 过采样会增加一些正类使得正、反类数目接近。
2. 过采样不能简单的对原始正类进行重复采样，否则会导致严重的过拟合。

   通常在原始正类之间插值来生成额外的正类。

3. 常见的有以下过采样策略：

- `SMOTE`方法：对于每个正类样本 $\vec{\mathbf{x}}_i^+$ ，从它的 k 近邻中随机选取一个样本点 $\hat{\vec{\mathbf{x}}}_i^+$ ，然后根据下式生成一个新的正类样本：$\vec{\mathbf{x}}_{new}^+=\vec{\mathbf{x}}_i^++(\hat{\vec{\mathbf{x}}}_i^+-\vec{\mathbf{x}}_i^+)\times\delta$ ，其中 $\delta\in[0,1]$ 是随机数。

  该方法有两个问题：

- 增加了正类样本之间重叠的可能性。
- 生成了一些没有提供有益信息的样本。
- `Borderline-SMOTE`方法：它类似`SMOTE`，但是对于每个正类样本 $\vec{\mathbf{x}}_i^+$ ，首先要评估：是否应该为该正类样本生成新的样本点。
  - 评估准则是：如果 $\vec{\mathbf{x}}_i^+$ 的 k 近邻中，有超过一半以上的反类样本，则为该正类样本生成新样本。

    > 反类样本超过一半，则说明该正类样本位于正类与反类的边界。如果 $\vec{\mathbf{x}}_i^+$ 的 k 近邻中正类样本比较多，则该正类样本很可能就是处于一群正类样本的内部。

  - 该方法生成新样本的算法与`SMOTE`方法相同。