来源:

https://blog.csdn.net/u011826404/article/details/72860607

上篇主要介绍了经典的降维方法与度量学习，首先从“维数灾难”导致的样本稀疏以及距离难计算两大难题出发，引出了降维的概念，即通过某种数学变换将原始高维空间转变到一个低维的子空间，接着分别介绍了kNN、MDS、PCA、KPCA以及两种经典的流形学习方法，k近邻算法的核心在于k值的选取以及距离的度量，MDS要求原始空间样本之间的距离在降维后的低维空间中得以保持，主成分分析试图找到一个低维超平面来表出原空间样本点，核化主成分分析先将样本点映射到高维空间，再在高维空间中使用线性降维的方法，从而解决了原空间样本非线性分布的情形，基于流形学习的降维则是一种“邻域保持”的思想，最后度量学习试图去学习出一个距离度量来等效降维的效果。本篇将讨论另一种常用方法–特征选择与稀疏学习。

12、特征选择与稀疏学习

最近在看论文的过程中，发现对于数据集行和列的叫法颇有不同，故在介绍本篇之前，决定先将最常用的术语罗列一二，以后再见到了不管它脚扑朔还是眼迷离就能一眼识破真身了~对于数据集中的一个对象及组成对象的零件元素：

> 统计学家常称它们为**观测**（**observation**）和**变量**（**variable**）；
>
> 数据库分析师则称其为**记录**（**record**）和**字段**（**field**）；
>
> 数据挖掘/机器学习学科的研究者则习惯把它们叫做**样本/示例**（**example/instance**）和**属性/特征**（**attribute/feature**）。

回归正题，在机器学习中特征选择是一个重要的“**数据预处理**”（**datapreprocessing**）过程，即试图从数据集的所有特征中挑选出与当前学习任务相关的特征子集，接着再利用数据子集来训练学习器；稀疏学习则是围绕着稀疏矩阵的优良性质，来完成相应的学习任务。

12.1 子集搜索与评价

一般地，我们可以用很多属性/特征来描述一个示例，例如对于一个人可以用性别、身高、体重、年龄、学历、专业、是否吃货等属性来描述，那现在想要训练出一个学习器来预测人的收入。根据生活经验易知：并不是所有的特征都与学习任务相关，例如年龄/学历/专业可能很大程度上影响了收入，身高/体重这些外貌属性也有较小的可能性影响收入，但像是否是一个地地道道的吃货这种属性就八杆子打不着了。因此我们只需要那些与学习任务紧密相关的特征，**特征选择便是从给定的特征集合中选出相关特征子集的过程。**

与上篇中降维技术有着异曲同工之处的是，特征选择也可以有效地解决维数灾难的难题。具体而言：**降维从一定程度起到了提炼优质低维属性和降噪的效果，特征选择则是直接剔除那些与学习任务无关的属性而选择出最佳特征子集。**若直接遍历所有特征子集，显然当维数过多时遭遇指数爆炸就行不通了；若采取从候选特征子集中不断迭代生成更优候选子集的方法，则时间复杂度大大减小。这时就涉及到了两个关键环节：**1.如何生成候选子集；2.如何评价候选子集的好坏**，这便是早期特征选择的常用方法。书本上介绍了贪心算法，分为三种策略：

> **前向搜索**：初始将每个特征当做一个候选特征子集，然后从当前所有的候选子集中选择出最佳的特征子集；接着在上一轮选出的特征子集中添加一个新的特征，同样地选出最佳特征子集；最后直至选不出比上一轮更好的特征子集。

> **后向搜索**：初始将所有特征作为一个候选特征子集；接着尝试去掉上一轮特征子集中的一个特征并选出当前最优的特征子集；最后直到选不出比上一轮更好的特征子集。

> **双向搜索**：将前向搜索与后向搜索结合起来，即在每一轮中既有添加操作也有剔除操作。

对于特征子集的评价，书中给出了一些想法及基于信息熵的方法。假设数据集的属性皆为离散属性，这样给定一个特征子集，便可以通过这个特征子集的取值将数据集合划分为V个子集。例如：A1={男,女}，A2={本科,硕士}就可以将原数据集划分为2*2=4个子集，其中每个子集的取值完全相同。这时我们就可以像决策树选择划分属性那样，通过计算信息增益来评价该属性子集的好坏。

$$\mathrm{Gain}(A) = \mathrm{Ent}(D) - \sum_{v=1}^{V} \frac{|D^v|}{|D|} \mathrm{Ent}(D^v)$$

信息增益越大
越有助于分类

$$\mathrm{Ent}(D) = -\sum_{i=1}^{|\mathcal{Y}|} p_k \log_2 p_k$$

此时，信息增益越大表示该属性子集包含有助于分类的特征越多，使用上述这种**子集搜索与子集评价相结合的机制，便可以得到特征选择方法**。值得一提的是若将前向搜索策略与信息增益结合在一起，与前面我们讲到的ID3决策树十分地相似。事实上，决策树也可以用于特征选择，树节点划分属性组成的集合便是选择出的特征子集。

12.2 过滤式选择（Relief）

过滤式方法是一种将特征选择与学习器训练相分离的特征选择技术，即首先将相关特征挑选出来，再使用选择出的数据子集来训练学习器。Relief是其中著名的代表性算法，它使用一个“**相关统计量**”来度量特征的重要性，该统计量是一个向量，其中每个分量代表着相应特征的重要性，因此我们最终可以根据这个统计量各个分量的大小来选择出合适的特征子集。

易知Relief算法的核心在于如何计算出该相关统计量。对于数据集中的每个样例xi，Relief首先找出与xi同类别的最近邻与不同类别的最近邻，分别称为**猜中近邻（near-hit）**与**猜错近邻（near-miss）**，接着便可以分别计算出相关统计量中的每个分量。对于j分量：

$$\delta^j = \sum_i -\mathrm{diff}(x_i^j, x_{i,\mathrm{nh}}^j)^2 + \mathrm{diff}(x_i^j, x_{i,\mathrm{nm}}^j)^2$$

直观上理解：对于猜中近邻，两者j属性的距离越小越好，对于猜错近邻，j属性距离越大越好。更一般地，若xi为离散属性，diff取海明距离，即相同取0，不同取1；若xi为连续属性，则diff为曼哈顿距离，即取差的绝对值。分别计算每个分量，最终取平均便得到了整个相关统计量。

标准的Relief算法只用于二分类问题，后续产生的拓展变体Relief-F则解决了多分类问题。对于j分量，新的计算公式如下：

$$\delta^j = \sum_i -\mathrm{diff}(x_i^j, x_{i,\mathrm{nh}}^j)^2 + \sum_{l \neq k} \left(p_l \times \mathrm{diff}(x_i^j, x_{i,l,\mathrm{nm}}^j)^2 \right)$$

其中pl表示第l类样本在数据集中所占的比例，易知两者的不同之处在于：**标准Relief 只有一个猜错近邻，而Relief-F有多个猜错近邻。**

12.3 包裹式选择（LVW）

与过滤式选择不同的是，包裹式选择将后续的学习器也考虑进来作为特征选择的评价准则。因此包裹式选择可以看作是为某种学习器**量身定做**的特征选择方法，由于在每一轮迭代中，包裹式选择都需要训练学习器，因此在获得较好性能的同时也产生了较大的开销。下面主要介绍一种经典的包裹式特征选择方法 – LVW（Las Vegas Wrapper），它在拉斯维加斯框架下使用随机策略来进行特征子集的搜索。拉斯维加斯？怎么听起来那么耳熟，不是那个声名显赫的赌场吗？歪果仁真会玩。怀着好奇科普一下，结果又顺带了一个赌场：

> **蒙特卡罗算法**：采样越多，越近似最优解，一定会给出解，但给出的解不一定是正确解；

> **拉斯维加斯算法**：采样越多，越有机会找到最优解，不一定会给出解，且给出的解一定是正确解。

举个例子，假如筐里有100个苹果，让我每次闭眼拿1个，挑出最大的。于是我随机拿1个，再随机拿1个跟它比，留下大的，再随机拿1个……我每拿一次，留下的苹果都至少不比上次的小。拿的次数越多，挑出的苹果就越大，但我除非拿

100次，否则无法肯定挑出了最大的。这个挑苹果的算法，就属于蒙特卡罗算法——尽量找较好的，但不保证是最好的。

而拉斯维加斯算法，则是另一种情况。假如有一把锁，给我100把钥匙，只有1把是对的。于是我每次随机拿1把钥匙去试，打不开就再换1把。我试的次数越多，打开（正确解）的机会就越大，但在打开之前，那些错的钥匙都是没有用的。这个试钥匙的算法，就是拉斯维加斯的——尽量找最好的，但不保证能找到。

LVW算法的具体流程如下所示，其中比较特别的是停止条件参数T的设置，即在每一轮寻找最优特征子集的过程中，若随机T次仍没找到，算法就会停止，从而保证了算法运行时间的可行性。

输入： 数据集 D;
特征集 A;
学习算法 $\mathfrak{L}$;
停止条件控制参数 T.

过程：

$E = \infty$;
$d = |A|$;
$A^* = A$;
$t = 0$;
while $t < T$ **do**
  随机产生特征子集 A';
  $d' = |A'|$;
  $E' = \text{CrossValidation}(\mathfrak{L}(D^{A'}))$;
  if $(E' < E) \vee ((E' = E) \wedge (d' < d))$ **then**
    $t = 0$;
    $E = E'$;
    $d = d'$;
    $A^* = A'$
  else
    $t = t + 1$
  end if
end while

输出： 特征子集 A^*

交叉验证
虽然精度没提升
但是特征数变少了
表示找到最优解
若T次都没找到，则不会给出解

12.4 嵌入式选择与正则化

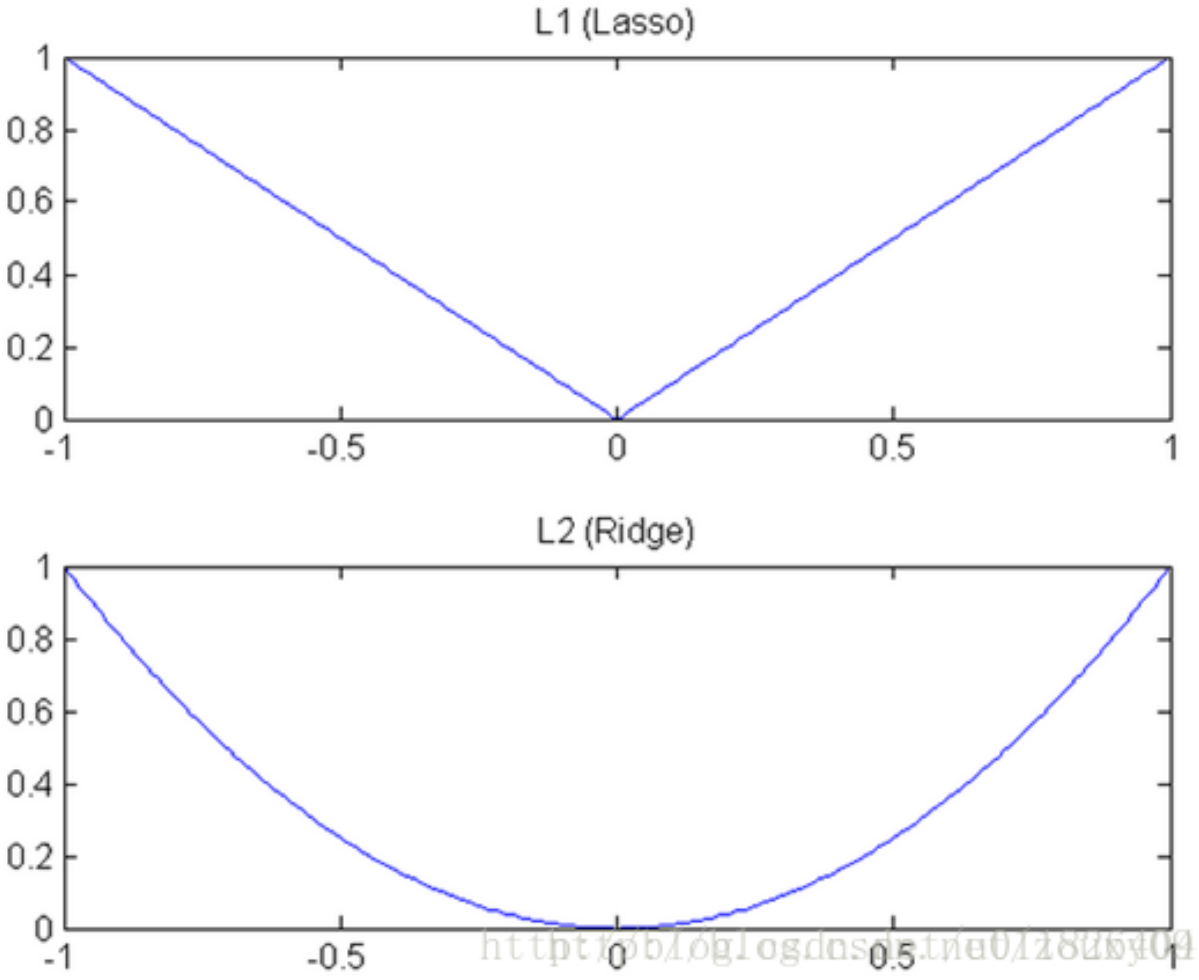
前面提到了的两种特征选择方法：**过滤式中特征选择与后续学习器完全分离，包裹式则是使用学习器作为特征选择的评价准则；嵌入式是一种将特征选择与学习器训练完全融合的特征选择方法，即将特征选择融入学习器的优化过程中**。在之前《经验风险与结构风险》中已经提到：经验风险指的是模型与训练数据的契合度，结构风险则是模型的复杂程度，机器学习的核心任务就是：**在模型简单的基础上保证模型的契合度**。例如：岭回归就是加上了L2范数的最小二乘法，有效地解决了奇异矩阵、过拟合等诸多问题，下面的嵌入式特征选择则是在损失函数后加上了L1范数。

$$\min_{\boldsymbol{w}} \sum_{i=1}^{m} (y_i - \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_i)^2 + \lambda \|\boldsymbol{w}\|_1 \ .$$

L1范数美名又约**Lasso Regularization**，指的是向量中每个元素的绝对值之和，这样在优化目标函数的过程中，就会使得w尽可能地小，在一定程度上起到了防止过拟合的作用，同时与L2范数（Ridge Regularization ）不同的是，L1范数会使得部分w变为0， 从而达到了特征选择的效果。

总的来说：**L1范数会趋向产生少量的特征，其他特征的权值都是0；L2会选择更多的特征，这些特征的权值都会接近于0**。这样L1范数在特征选择上就十分有用，而L2范数则具备较强的控制过拟合能力。可以从下面两个方面来理解：

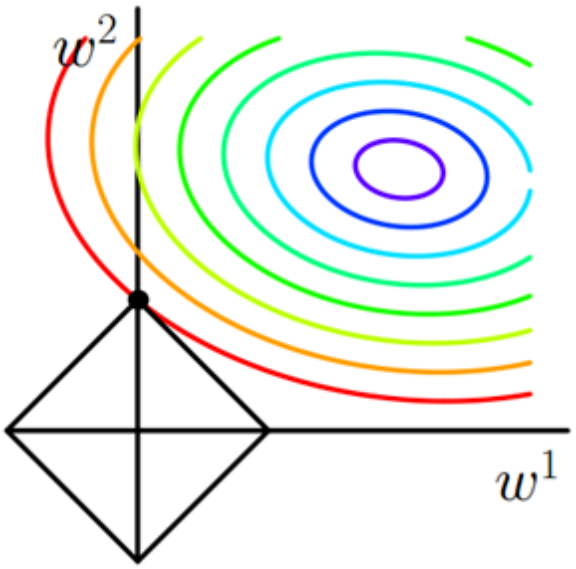
（1）**下降速度**：L1范数按照绝对值函数来下降，L2范数按照二次函数来下降。因此在0附近，L1范数的下降速度大于L2范数，故L1范数能很快地下降到0，而L2范数在0附近的下降速度非常慢，因此较大可能收敛在0的附近。



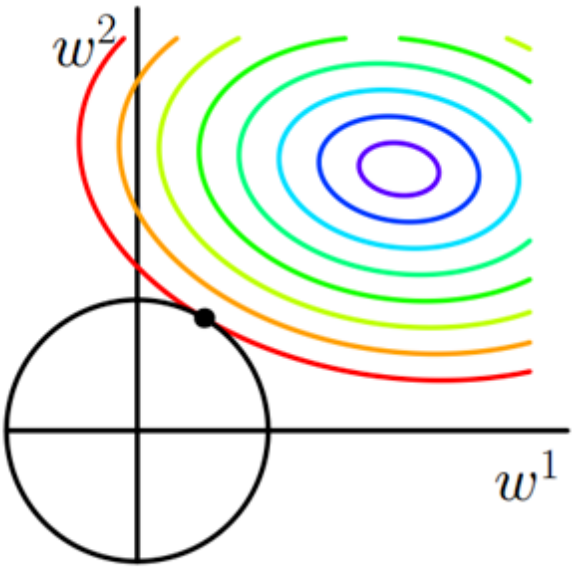
（2）**空间限制**：L1范数与L2范数都试图在最小化损失函数的同时，让权值W也尽可能地小。我们可以将原优化问题看做为下面的问题，即让后面的规则则都小于某个阈值。这样从图中可以看出：L1范数相比L2范数更容易得到稀疏解。

$$Lasso: \quad \min_w \frac{1}{n} \| y - X w \|^2, \quad s.t. \| w \|_1 \le C$$

$$Ridge: \quad \min_w \frac{1}{n} \| y - X w \|^2, \quad s.t. \| w \|_2 \le C$$



(a) ℓ_1-ball meets quadratic function. ℓ_1-ball has corners. It's very likely that the meet-point is at one of the corners.



(b) ℓ_2-ball meets quadratic function. ℓ_2-ball has no corner. It is very unlikely that the meet-point is on any of axes.

12.5 稀疏表示与字典学习

当样本数据是一个稀疏矩阵时，对学习任务来说会有不少的好处，例如很多问题变得线性可分，储存更为高效等。这便是稀疏表示与字典学习的基本出发点。稀疏矩阵即矩阵的每一行/列中都包含了大量的零元素，且这些零元素没有出现在同一行/列，对于一个给定的稠密矩阵，若我们能**通过某种方法找到其合适的稀疏表示**，则可以使得学习任务更加简单高效，我们称之为**稀疏编码（sparse coding）**或**字典学习（dictionary learning）**。

给定一个数据集，字典学习/稀疏编码指的便是通过一个字典将原数据转化为稀疏表示，因此最终的目标就是求得字典矩阵B及稀疏表示α，书中使用变量交替优化的策略能较好地求得解，深感陷进去短时间无法自拔，故先不进行深入...

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

xi : d*1
B : d*k字典矩阵
ai : k*1，xi的稀疏表示

使得ai尽可能稀疏

12.6 压缩感知

压缩感知在前些年也是风风火火，与特征选择、稀疏表示不同的是：它关注的是通过欠采样信息来恢复全部信息。在实际问题中，为了方便传输和存储，我们一般将数字信息进行压缩，这样就有可能损失部分信息，如何根据已有的信息来重构出全部信号，这便是压缩感知的来历，压缩感知的前提是已知的信息具有稀疏表示。下面是关于压缩感知的一些背景：

- 压缩感知最初提出时，是针对稀疏信号x，给出观测模型y=Φ*x时，要有怎么样的Φ，通过什么样的方式可以从y中恢复出x。（PS：稀疏信号，是指在这个信号x中非零元素的个数远小于其中零元素的个数。）
- 然而，很多信号本身并非稀疏的，比如图像信号。此时可以通过正交变换Ψ'，将信号投影到另外一个空间，而在这个空间中，信号a=Ψ'*x(analysis model)变得稀疏了。然后我们可以由模型y=Φ*a，即y=Φ*Ψ'*x，来恢复原始信号x。
- 后来，人们发现不仅仅能够通过正交变换，得到稀疏的信号；还可以通过一个字典D，得到稀疏信号x=D*a(synthesis model)，a是稀疏的，为了增强变换后信号的稀疏性，通常D是过完备的。即模型y=Φ*x=Φ*D*a，此时记A^{CS}=Φ*D，即为**感知矩阵**。这个模型，是我们现在最常用的。

在此，特征选择与稀疏学习就介绍完毕。在很多实际情形中，选了好的特征比选了好的模型更为重要，这也是为什么厉害的大牛能够很快地得出一些结论的原因，谓：吾昨晚夜观天象，星象云是否吃货乃无用也~