Machine Learning in Action `The Book`

> 2018年7月4日 - 2018年8月27日

`k-近邻算法` `决策树` `朴素贝叶斯` `Logistic回归` `SVM` `AdaBoost` `线性回归` `树回归` `K-均值聚类` `Apriori算法` `FP-Growth算法` `PCA` `SVD` `大数据与MapReduce`

k-近邻算法

算法描述

k-近邻分类算法：通过计算预测点与所有训练点的距离，排序取出最近的k个训练点的标签，投票决定预测点的标签。

算法核心

1. **训练点搜索方法**：也就是训练点的数据结构。该算法适合应用于小数据集，当样本量m和特征量n都很大时，会引发维度灾难。所以使用有效的数据结构存储训练点可以有效的减少距离计算量，基本思想是如果A和B离得很远，B和C离得很近，那么A和C的距离就不需要明确计算。SKLearn中使用Ball Tree和KD Tree处理这个问题。
2. **距离的计算方法**：书中使用欧拉距离，实际上可以使用任何描述两点间关系的方法进行计算，比如每个特征分别距离多少个标准差，这些标准差的统计参数。
3. **超参数k的取值**：k代表基于每个预测点的k个最近邻，是整数值。k值越小，学习的近似误差(approximation error)越小，估计误差(estimation error)越大，反之则相反。随k的增长，决策边界会越来越平滑。实际上还可以用基于每个预测点的固定半径r内的邻居数量进行计算，固定半径r可以是浮点数。SKLearn里使用sklearn.neighbors模块中的KNeighborsClassifier和RadiusNeighborsClassifier实现。
4. **归一化**：由于每个特征的空间不一样，数量级大的特征对距离计算的影响也大，所以需要要进行归一化处理，以实现所有特征在同一比例空间，保证权重的统一。书中的方法是 $\frac{x-min}{max-min}$ ，SKLearn使用sklearn.preprocessing模块中的MinMaxScaler和MaxAbsScaler实现。
5. **排序和投票方法**：书里面采用统一权重比多少，实际投票时可以根据距离的大小计算权重，使得附近点对于投票所作出的贡献多于远处点。

算法过程：

遍历所有测试点，计算预测点和所有测试点的距离 → 对所有距离进行排序，取得前k名的标签 → 对得到的标签按分类进行计数，取数量最多的一个标签作为结果

算法特点

- 优点：准确度较高，对异常值不敏感，可以处理多分类问题
- 缺点：计算复杂度高，空间复杂度高

机器学习实战

1. k-近邻分类算法
2. 从文本文件中解析和导入数据
3. 使用Matplotlib创建扩散图
4. 归一化数值

决策树

算法描述

ID3决策树：通过计算子集的信息增益（熵）找到划分数据集的最优特征（遍历），并基于最优特征划分数据集，对子集递归上述过程，直到叶节点达到最高纯度，或者数据集不可再分。

算法核心

1. **熵的计算**：熵 = Entropy，计算方法是

$$E = -\sum_{k=1}^{K}(p_k log_2(p_k))$$

其中p_k为数据集（数据子集）中第k类样本的标签比例

2. **信息增益的计算**：信息增益 = Info Gain，计算方法是

$$Info\ Gain = E_0 - \sum_{n=1}^{N}(P_n \cdot E_n)$$

其中E_0为数据集的熵，E_n为子集的熵，求和前需要乘以该子集在数据集中的样本量占比P_n

3. **寻找最优划分特征**：使用所有特征依次划分数据集，通过计算数据集和子集的信息增益，选择信息增益最大的特征。
4. **划分子集**：选择最优特征，根据特征值的不同将数据集划分为不同子集。所有子集将不含划分特征，即该特征被丢弃，但不同树分支丢弃的时机不同（不同树分支在同一层有可能选用不同的最优划分特征）。
5. **递归和终止条件**：通过递归对根节点和每一个分支节点进行寻找最优特征和划分处理。递归的结束条件是子集的熵为0（子集的纯度达到最高）；或者子集只剩下一个特征，通过计算不同标签的数量，取数量最多的标签为结果。
6. **进行分类**：找到代表根节点的字典key和value（此时的value为下一层字典），通过字典key找到预测数据对应的列，使用列值和下一层字典的key依次比较，如果值相等，则判断下一层字典的value类型，如果是字典则进行递归（相当于走了这一条树分支），否则返回下一层字典的value（抵达叶节点）。

算法过程

决策树构建过程：计算数据集的熵 → 遍历所有特征，根据每个特征的值划分数据集 → 基于子集的标签计算熵，并对该划分下的所有子集熵进行加权求和 → 比较数据集和子集的熵差，得到信息增益，取信息增益最大的特征作为划分方法，并在子集中丢弃这个特征 → 递归进行上述步骤，直到子集的熵为0，或者子集只有一个特征，通过投票选取标签 → 完成决策树构建

预测数据分类过程：找到字典第一个key，value → 通过key找到预测数据的对应列 → 使用列值和value字典的key做比较，找到下一层字典对应的key → 判断下一层字典对应key的value类型 → value类型为字典则进行递归，否则返回value

算法特点

- 优点：计算速度快，结果可以被解释；树结构可以是多分支、多分类
- 缺点：只能处理离散型数据，容易过拟合，对缺失值敏感

机器学习实战

1. 决策树简介
2. 在数据集中度量一致性
3. 使用递归构造决策树
4. 使用Matplotlib绘制树形图

朴素贝叶斯

算法描述

朴素贝叶斯分类算法：根据历史数据集的特征出现概率和类别先验概率，结合当前样本的特征出现情况，计算多个类别的条件概率（后验概率），并做大小比较。

算法核心

1. **条件概率**：贝叶斯公式 = Bayes' theorem，计算公式是

$$P(C_i|W_n) = \frac{P(W_n|C_i)P(C_i)}{P(W_n)}$$

其中$P(C_i)$为C_i类别的先验概率

$P(C_i|W_n)$是C_i类别的后验概率，表示在W_n事件发生的情况下，C_i类别发生的概率

$\frac{P(W_n|C_i)}{P(W_n)}$为调整因子，表示$W_n$事件发生对$C_i$类别发生的影响程度，大于$1$表示增强

2. **独立事件假设**：由于 W_n 事件的出现条件相互独立，所以条件概率公式中的分子 $P(W_n|C_i) = P(W_1|C_i)P(W_2|C_i)\cdots P(W_n|C_i)$，这里有个问题就是如果 W_n 在某个类别的历史数据中没有出现，那么在进行分类时会出现和0相乘的情况，这里需要用到初始值假设；同时分母 $P(W_n) = P(W_1)P(W_2)\cdots P(W_n)$，但由于比较大小时，$P(W_n)$ 的值是一样的，所以经常忽略计算。

3. **初始值假设**：书中为避免出现 $P(W_n|C_i)$ 相乘得0的情况，将 $P(W_n|C_i)$ 的分子初始化为1，即假设 W_n 在每个类别 C_i 中都至少出现1次，将分母初始化为2，最终假设特征 W_n 在不同 C_i 类别中都会出现，且历史出现概率为50%（这里可以不按书中的方法，直接进行对矩阵进行初始化负值）；Paul Graham的方法有2个假设，一是假如 W_n 只在某一类别中出现，那么它在其他类别中的出现概率为1%；如果 W_n 没出现在历史词库中，但出现在了预测样本中，那么它的初始 $P(W_n|C_i)$ 为40%。
4. **词集模型和词袋模型**：词集模型（词库模型）将每个事件 W_n 出现与否作为一个特征，特征值只有1和0；词袋模型将每个事件的出现次数作为一个特征，特征值包括0~n。
5. **词频-逆文档频率**：TF-IDF = Term Frequency-Inverse Document Frequency。用于衡量一个词对一个语料库中的一篇文档的代表程度，一个词语在一篇文章中出现次数越多, 同时在所有文档中出现次数越少, 越能够代表文档。

$$TF\text{（词频）} = \frac{\text{某个词在文档中的出现次数}}{\text{文档总词数}} \quad or \quad \frac{\text{某个词在文档中的出现次数}}{\text{文档中出现次数最多的词的出现次数}}$$

$$IDF\text{（逆文档频率）} = log\frac{\text{语料库的文档总数}}{\text{包含该词的文档数}+1} \quad or \quad log\frac{N+1}{N(w)+1}+1$$

$$TFIDF = TF \cdot IDF$$

6. **乘法运算到对数运算的转换**：由于 $P(W_n|C_i)$ 可能很小，会造成计算的下溢出，所以使用 $log(ab) = log(a) + log(b)$，将对数运算替换为乘法运算。
7. **进行分类（新样本如何调用历史数据）**：计算新样本中的 W_n 出现次数（词袋模型，不出现的为0，即不参与后面的计算），基于Numpy广播与历史 $P(W_n|C_i)$ 相乘，基于独立事件假设计算 $P(W_1|C_i)P(W_2|C_i)\cdots P(W_n|C_i)$，最终和历史 $P(C_i)$ 相乘，得到 C_i 类别的条件概率的分子。
8. **和Paul Graham方法的对比**：Paul Graham的分类方法只计算了单个类别 S_i 的条件概率，并用这个概率和90%做对比，不是对不同 C_i 的条件概率进行对比；不同类别 C_i 的词库是是一样的，对于特定 W_1 只在 C_1 中出现的情况采用假设；初始 $P(W_n|C_i)$ 值考虑了2种情况，因而有2种假设，不是书中统一的50%；由于不是不同 C_i 的条件概率的对比，所以需要计算出最终的条件概率，即计算完整的 $\frac{P(W_n|C_i)P(C_i)}{P(W_n)}$，这里用到联合概率公式，将计算变化为 $\frac{P_1P_2}{(P_1P_2+(1-P_1)(1-P_2))}$，其中 $P_1 = P(S|W_1)$，即需要先算出单个 S_i 在 W_n 出现下的条件概率；每个预测样本选取 $P(S|W)$ 最高的15个词参加计算。
9. **交叉留存验证（Hold-out Cross Validation）**：将数据集按照一定比例（ $\frac{1}{5}$ 等），随机划分为训练集和测试集，循环进行上述步骤，求error rate的平均值作为结果。

算法过程

根据历史数据集计算词库 → 根据历史数据集计算 W_n 的词库向量（循环合成为词库矩阵）和不同类别的样本特征值总数 → 针对不同类别，对词库矩阵中对应样本的词库向量进行求和，计算 $P(W_n|C_i)$ 数列和 $P(C_i)$ 数值 → 计算新样本的词库向量 → 将新样本的词库向量和 $P(W_n|C_i)$ 数列相乘，取得出现特征的概率，互相之间再进行相乘，再与 $P(C_i)$ 数值相乘，得到条件概率公式的分子 → 进行条件概率的比较

算法特点

- 优点：在数据较少的情况下依然有效（基于假设）；可以处理多特征、多分类问题
- 缺点：只能处理离散型数据

机器学习实战

1. 使用概率分布进行分类
2. 学习朴素贝叶斯分类器
3. 解析RSS源数据
4. 使用朴素贝叶斯来分析不同地区的态度

Logistic回归

算法描述

逻辑回归：通过梯度下降法找到代价函数的最小值，从而得到Sigmoid函数的最佳回归系数，将预测点带入Sigmoid函数的结果和0.5比较，得出分类。

算法的核心

1. **Sigmoid函数**：$g(x)=\frac{1}{(1+e^{-x})}$ 。当 $x>=0$ 时，$0.5<=g(x)<1$；当 $x<0$ 时，$0<g(x)<0.5$ 。
2. **假设函数**：Hypothesis = $h_\theta(x)$ 。在Logistic回归中 $g(z)=\frac{1}{(1+e^{-z})}$ ，其中 $z=\theta^T x$ ，所以 $h_\theta(x)=g(z)=g(\theta_0+\theta_1x_1+\cdots+\theta_nx_n)$ ，而且 $g(z)=Sigmoid(z)=Sigmoid(\theta_0+\theta_1x_1+\cdots+\theta_nx_n)$ ，表示在参数为 θ 的条件下，将 x 带入Sigmoid函数时，可以得到对应的 y 值（0或1）。
3. **代价函数**：Cost Function = $J(\theta)$ 。一般选用预测函数 $h_\theta(x)$ 和 y 的误差汇总（需要是凸型函数，意义上可以衡量误差）

线性回归里使用的是均方误差，计算公式是

$$J(\theta)=\frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^i)-y^i)^2$$

$$Logistic回归中使用的是J(\theta)=\begin{cases}当y=1时 & -log(h_\theta(x))\\ 当y=0时 & -log(1-h_\theta(x))\end{cases}$$

$$合并可以得到J(\theta)=-\frac{1}{m}\sum_{i=1}^{m}y^i log(h_\theta(x^i))+(1-y^i)log(1-h_\theta(x^i))$$

3. **批量梯度下降法（BGB）**：当预测函数 $h_\theta(x)$ 和 y 的误差汇总，也就是 $J(\theta)$ 达到最小值时，拟合成功，所以需要用梯度下降的办法求 $J(\theta)$ 最小时的回归参数 θ 。θ 的更新过程为 $\theta:=\theta_j-a\frac{\partial J(\theta_j)}{\partial\theta_j}$ ，a 为学习步长（学习率），$\frac{\partial J(\theta_j)}{\partial\theta_j}$ 为 $J(\theta_j)$ 在 θ_j 方向上的偏导数，所以 $\frac{\partial J(\theta_j)}{\partial\theta_j}$ 表证的是 $J(\theta)$ 的最速下降方向（梯度）

针对代价函数求偏导数，可得$\theta_j := \theta_j - a\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^i)-y^i)x_j^i$

书中公式为$\theta_j := \theta_j + a\frac{1}{m}\sum_{i=1}^{m}(y^i-h_\theta(x^i))x_j^i$

由于$a\frac{1}{m}$为常数，所以可以合并为a

不同的公式只是将负号提出来而已，这代表梯度下降和梯度上升本质上是一样的

4. **梯度计算向量化**：已知数据集，标签，θ 的矩阵形式如下

$$X=\begin{bmatrix}x^1\\x^2\\\cdots\\x^m\end{bmatrix}=\begin{bmatrix}x_0^1 & x_1^1 & \cdots & x_n^1\\x_0^2 & x_1^2 & \cdots & x_n^2\\\cdots & \cdots & \cdots & \cdots\\x_0^m & x_1^m & \cdots & x_n^m\end{bmatrix}\quad y=\begin{bmatrix}y^1\\y^2\\\cdots\\y^m\end{bmatrix}\quad \theta=\begin{bmatrix}\theta^1\\\theta^2\\\cdots\\\theta^m\end{bmatrix}$$

可以得到向量化后，θ的计算步骤为

$$(1)\quad A=X\theta$$
$$(2)\quad E=g(A)-y$$

当$J=0$时，θ的的更新过程为

$$\begin{aligned}\theta_0 &:= \theta_0 - a\sum_{i=1}^{m}(h_\theta(x^i)-y^i)x_0^i\\&=\theta_0-a\sum_{i=1}^{m}e^ix_0^i\\&=\theta_0-a\begin{bmatrix}x_0^1 & x_0^2 & \cdots & x_0^m\end{bmatrix}E\end{aligned}$$

当J为任意数值时，$\theta_j := \theta_j - a\begin{bmatrix}x_j^1 & x_j^2 & \cdots & x_j^m\end{bmatrix}E$

堆叠起来，可以得到

$$(3)\quad \theta := \theta - aX^TE$$

5. **随机梯度下降法（SGB）**：批量梯度下降法每次更新回归系数 θ 都需要计算整个数据集，计算复杂度太高，所以改进方法是一次仅随机选取一个样本点来更新回归系数，即在线学习。代码上的改进为θ，error，sigmoid计算结果都为数值，而非矩阵。但有效减少计算量的结果是分类准确度的下降。

算法过程

将Sigmoid函数表示为 $g(z)$ → 选择代价函数 $J(\theta)$ 的形式（书中根据最大似然估计得到） → 计算 $J(\theta)$ 的偏导数，得到 θ 的更新公式 → 根据设定的迭代次数和学习率，循环更新 θ，实现梯度下降，最终返回 θ → 将预测点带入 $g_\theta(z)$ 的结果和0.5做比较，大于0.5表示预测为1

算法特点

- 优点：算法过程易于实现（梯地下降法的数据需要标准化）
- 缺点：只能二分类；批量梯度下降法准确度高，但速度慢，计算复杂度高；随机梯度下降法会造成欠拟合，分类准确度可能不高

机器学习实战

1. Sigmoid函数和Logistic回归分类器
2. 最优化理论初步
3. 梯度下降最优化算法
4. 数据中的缺失项处理

AdaBoost算法

算法描述

AdaBoost算法：串行训练多个弱分类器，基于每次迭代的错误率不断更新样本权重和分类器权重，并通过对前面所有分类器的预测结果进行加权计算得到强分类器。

算法核心

1. **自适应增强**：AdaBoost = Adaptive Boost。每个新弱分类器都根据上一个分类器的性能进行训练，并设定样本权重和弱分类器权重。针对样本权重，前一个分类器中被正确分类的样本的权重会减小，被错误分类的样本的权重会增强，权值更新过的训练集被用于训练下一个弱分类器；针对弱分类器权重，会参与到最终强分类器的结果的加权计算中。
2. **符号定义**：

D_t：第t轮训练集的权重分布

w_t^i：第i号训练样本点在第t轮的权重

$h_t(x)$：第t轮的弱分类器（t也可以视为弱分类器的编号）

$H(x)$：由每个$h(x)$预测结果加权求和得到的强分类器

e_t：第t轮的错误率

a_t：第t轮新生成的弱分类器的权重（t也可以视为弱分类器的编号）

3. **弱分类器的构建**：书中使用单层决策树作为弱分类器，单层决策树的特点是每次只使用一个特征作为分类依据，在给定训练集中寻找在当前样本权重下最优弱分类器的过程是：遍历所有特征，以一定步长遍历该特征从max到min的所有可能取值，以该取值作为阈值，遍历该阈值两侧各为目标类别的2种情况（如左侧为正/负），对比预测值和实际标签的差异得到错误率，取3层循环中错误率最下的弱分类器（弱分类器标识包括：特征index、阈值、正类在阈值的哪一侧）。
4. **错误率的计算**：错误率为被弱分类器错误分类的样本权重之和，计算方法是

$$e_t = \sum_{i=1}^{N} w_i(h_t(x_i) \neq y_i)$$

5. **样本权重的计算**：始化权重为 $D_0 = (w_1, w_2 \dots w_n) = (\frac{1}{N} \dots \frac{1}{N})$，更新的样本权重为

$$D_{t+1}(i) = \frac{D_t(i)exp(-a_t y_i h_t(x_i))}{Z_t}$$，其中$Z_t = 2\sqrt{e_t(1-e_t)}$

经过推导，可以得到$D_{t+1}(i) = \begin{cases} \text{错误分类的样本} & \frac{D_t(i)}{2e_t} \\ \text{正确分类的样本} & \frac{D_t(i)}{2(1-e_t)} \end{cases}$

6. **弱分类器权重的计算**：计算该分类器所预测的结果在最终计算时的比重，计算方法是

$$a_t = \frac{1}{2} ln(\frac{1-e_t}{e_t})$$

7. **强分类器的组合**：按弱分类器权重 a_t 组合各个预测结果，可得

$$H(x) = \sum_{t=1}^{T} a_t h_t(x)$$

通过符号函数$sign$的作用，得到最终结果$H_{final} = sign(H(x))$

算法过程

强分类器构建过程：初始化样本权重为 $\frac{1}{N}$ → 定义弱分类器 $h(x)$ → 将 D_t 和训练集带入 $h(x)$，得到错误率 e_t → 通过 e_t 计算弱分类器的权值 a_t，更新样本权值 D_{t+1} → 迭代上述过程，直到达到整体错误率为0或预定最大迭代次数

分类过程：遍历每个弱分类器 → 基于弱分类器得到预测结果（弱分类器标识包括：特征index、阈值、正类在阈值的哪一侧） → 将所有弱分类器的预测结果乘以弱分类器权重并求和，通过sign函数得到最终结果

算法特点

- 优点：泛化能力强，大部分分类器都可以作为弱分类器使用，无参数调整。
- 缺点：对离群点敏感，离群点的样本权重会不断提高。

机器学习实战

1. 组合相似的分类器来提高分类性能
2. 应用AdaBoost算法：弱分类器的生成数量（算法的迭代次数）和偏差有关，数量过容易造成过拟合。
3. 处理非均衡分类问题：
    - 书中提到一个数据集中，不仅类别数量可能差异很大，分类的代价也并不相同。以垃圾邮件的判别为例：TP表示成功将垃圾邮件识别（丢进垃圾桶）；TN表示成功将正常邮件识别；FP表示误删正常邮件；FN表示漏判垃圾邮件，这里FP比FN的代价要高。但在癌症诊断案例中，则情愿误判也不漏判。
    - 样本的不均衡可以通过过采样（随机重复抽取样本点）和欠采样（删除样本点）将数据集改造为新的形式（有代价）。
    - 考察不均衡数据集分类性能的评价方法：Precision，Recall，ROC，AUC。

线性回归

算法描述

线性回归：对线性可分数据集构建回归方程（特征数量决定回归方程项数），基于最小二乘法得到代价函数，使代价函数求导为0，进而通过最优化方法或正规方程法得到最佳回归系数。

算法核心

1. **最小二乘法**：线性回归的代价函数，用于计算回归方程和 y 的最小均方误差，计算公式是

$$MSE = \frac{1}{2m}\sum_{i=1}^{m}(y_i - x_i^T w)^2$$

回归方程为$y = x^T w = x_0 w_0 + x_1 w_1 + \cdots + x_n w_n$，其中$x_0 = 1$

2. **正规方程**：正规方程法和梯度下降法的代价函数是一样的，将向量表达式转为矩阵表达式

$$\begin{aligned} J(\theta) &= \frac{1}{2m}(X\theta - y)^2 \\ &= \frac{1}{2m}(X\theta - y)^T(X\theta - y) \end{aligned}$$

使$\frac{\partial J(\theta)}{\partial \theta} = 0$，可得$\theta = (X^T X)^{-1} X^T y$

正规方程法的使用条件是$X^T X$可逆

3. **局部加权线性回归**：为预测点附近的点赋予一定的权重，计算公式是

W为单位矩阵，$w(i,i) = exp(\frac{|x_i - x|}{-2k^2})$

$$\theta = (X^T W X)^{-1} X^T W y$$

4. **岭回归**：当特征很多的时候，线性回归模型容易过拟合，此时需要通过正则化方法缩减系数，正则化线性回归的代价函数是

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^i) - y^i)^2 + \lambda\sum_{j=1}^{n}\theta^2$$

I为单位矩阵，$\theta = (X^T X + \lambda I)^{-1} X^T y$

λ为正则化参数，用于限定所有回归系数的平方和，即$\sum_{j=1}^{n}\theta_i^2 \leq \lambda$

5. **lasso回归**：和岭回归一样，lasso回归也对回归系数做了限定，对应的约束条件是

$$\sum_{j=1}^{n}|\theta_i| \leq \lambda$$

6. **前向逐步回归**：一种贪心算法，需要将数据集正态化（满足均值为0和单位方差），并将所有回归系数都设为1，在每轮迭代中遍历每个特征，对每个特征增大或减少固定步长，得到当前最佳回归系数。

算法过程

将数据集转化为矩阵 $\rightarrow$ 将矩阵带入回归系数计算公式，得到 $\boldsymbol{w}$

算法特点

- 优点：结果易于理解，正规公式计算不复杂（正规公式法的数据不需要标准化）
- 缺点：对非线性数据拟合不够好

机器学习实战

1. 线性回归
2. 局部加权线性回归
3. 岭回归和逐步线性回归
4. 预测鲍鱼年龄和玩具价格

树回归

算法描述

CART算法：使用二分法对数据进行划分，划分的方法是遍历所有特征，遍历所有特征值，通过对比划分前后的误差找到最优划分方法，直到达到划分的停止条件，返回业叶节点创建函数的结果。

算法核心

1. **CART**：CART = Classification And Regression Trees。分类回归树使用二元切分法，在无法使用全局模型拟合复杂数据的情况下，将数据集切分为许多易建模的数据子集。根据叶节点的不同分为回归树（常数）和模型树（线性方差），但两者用于衡量划分是否有效的误差计算方法都是总方差。
2. **CART树的存储**：由于CART树是二分树，所以树字典可以固定数据结构。字典包括4个元素：划分特征idnex，划分特征阈值，左子树，右子树。子树的值有3种：常数，线性模型的参数，树。
3. **数据集的切分过程**：给定切分的特征index和特征阈值，数据集中该特征值比阈值小的划分为左子集，比阈值大的划分为右子集。
4. **寻找最优划分**：遍历每个特征，遍历每个特征值，将数据集划分为2份后计算误差（对比划分前方差总和以及划分后2个数据子集的方差总和求和），取误差最小的划分方法，返回最优特征index和特征阈值。当满足停止条件时（误差减少量，子集样本量，子集标签值是否唯一），不再划分并返回叶节点（特征index为None，特征阈值更改为叶节点生成函数的返回值）。
5. **树的构建过程**：构建树的过程是递归进行（1）寻找最佳划分；（2）划分数据集。返回枝节点时，生成字典元素（包括最优特征index，特征阈值，左子集递归，右子集递归）。返回叶节点时（通过最优特征index为None判断），停止递归并返回特征阈值。
6. **主要程序函数**：包括5个主要函数，分别是：树构建函数，寻找最佳划分函数，数据集划分函数，误差测量函数，叶节点生成函数。

7. **回归树的构建**：回归树在每个叶节点上使用数据子集的标签均值做预测。叶节点生成函数用于求解子集平均值（返回均值）。误差测量函数用于计算方差总和（计算方法是 $var \cdot m$ ）。
8. **模型树的构建**：模型树在每个叶节点上使用一个线性模型做预测。叶节点生成函数用于求解线性回归模型（返回回归系数）。误差测量函数用于基于线性回归计算方差总和。
9. **和ID3树的对比**：ID3树无法处理连续型数据；树可以有多个分枝；在使用某个特征切分数据集后，该特征在之后的算法过程中将被丢弃。
10. **前剪枝**：在树的枝节点创建之前进行，从上到下决定是否进行数据集划分的依据有：该划分下子集的误差减少程度，子集样本量，子集标签值的纯度等。
11. **后剪枝**：在使用训练集完成树的创建后进行，使用测试集决定枝节点的划分是否有效，从下到上决定是否合并叶节点的方法是：按树结构向下划分测试集，当抵达下一层是叶节点的枝节点时，对比枝节点的方差和和下一层相邻2个叶节点的方差总和求和，如果误差减少程度不大，则将枝节点变成叶节点（枝节点不划分，直接返回子集均值或回归系数），然后一层一层往上递归。
12. **使用相关系数评价模型**：计算预测值和真实标签值的相关系数，即 R^2 ，值越接近1越好。

算法过程

树构建过程：遍历所有特征，遍历所有特征值，通过计算划分前后的误差，找到最佳划分方法 → 使用字典存储枝节点 → 对枝节点存储的2个下一层节点递归进行树创建过程 → 当子集无法继续划分时，返回叶节点生成函数的值 → 根据子集划分的结果判断是否到达递归停止条件，如果是，将枝节点存储的下一层节点的值设置为叶节点生成函数的返回值

回归过程：设置需要调用的误差测量函数和叶节点生成函数，选择回归树还或模型树 → 输入预测点，根据字典树的最优特征index和特征阈值，决定预测点向哪个方向的树枝前进 → 递归进行上升过程，递归的参数包括预测点的和枝节点（看成一棵树），直到抵达叶节点

算法特点

- 优点：可以处理非线性数据；算法灵活，可以应用于回归或分类，叶节点可以是常数或模型；可以通过树剪枝减轻过拟合
- 缺点：树只能是二分树

机器学习实战

1. CART算法
2. 回归与模型树
3. 树剪枝算法
4. Python中GUI的使用

K-均值聚类

算法描述

K-均值聚类算法：通过计算数据点和质心的距离，决定数据点的聚类标签，在每轮聚类完成后，使用簇的特征均值作为新的质心，再次进行距离计算，直到每个数据点的聚类标签不再改变。

算法核心

1. **首轮质心的生成**：随机生成K个质心，质心的特征取值范围在数据集每个特征的上下限之内。
2. **质心的更新与停止**：每轮聚类完成后，都重新计算每个簇的质心，将簇的每个特征的均值作为质心的取值。重复该过程，直到在某一轮聚类过程中，所有数据点的聚类标签都不再发生改变（即计算任意数据点和所有质心的距离时，最小距离依据是和当前质心的距离）。
3. **误差平方和的计算**：SSE = Sum of Squared Error。误差平方和在K-均值聚类中被定义为簇中所有数据点到质心的距离平方求和，用于衡量聚类效果，表示簇中的点接近质心的程度。聚类的目标是在保持簇数目不变的情况下减少SSE。
4. **聚类后处理办法**：后处理的2种办法：（1）计算所有质心之间的距离，合并距离最近的2个质心；（2）取任意2个簇的组合，对比这两个簇合并前后的SSE，合并SSE减少量较大的组合。
5. **二分K-均值聚类中的划分**：当簇数量小于K时，遍历已有的每个簇，对每个簇进行K=2的K-均值聚类，计算划分前后的SSE减少量，选择SSE减少最多的划分方法；或者直接对SSE最大的簇进行2分。

算法过程

K-均值聚类过程：随机生成K个质心，质心的特征取值范围在数据集每个特征的上下限之内 → 遍历每个数据点，遍历每个质心，计算数据点和质心的距离，取距离最小的质心index为聚类标签 → 首轮聚类完成后，对每个簇的所有特征计算均值，以每个特征的均值作为新质心的特征取值 → 重复上述过程，直到在计算任意数据点和所有质心的距离时，得到的聚类标签没有发生改变

二分K-均值聚类过程：将所有数据点视为同一个簇，取该簇所有特征的均值作为首个质心 → 当簇数量小于K时，遍历当前已有的簇，对每个簇进行K=2的K-均值聚类，计算划分前后的SSE减少量，选择SSE减少最多的划分方法 → 更新训练点的聚类标签，更新质心数组 → 重复上述过程，直到簇的数量等于K

算法特点

- 优点：算法过程简单
- 缺点：首轮质心的生成是随机的，容易得到局部最优结果；在大数据集上计算较慢

机器学习实战

1. K-均值聚类算法
2. 对聚类得到的簇进行后处理
3. 二分K-均值聚类算法
4. 对地理位置进行聚类

Apriori算法

算法描述

Apriori算法：基于Apriori原理，在发现频繁项集的过程中，小于最小支持度的项集将不再参与下一轮的组合；在发现关联规则的过程中，小于最小可信度的规则右部将不再参与下一轮的组合，以此减少计算量。

算法核心

1. **关联分析**：在数据集中寻找项目间的隐含关系被称作关联分析（association analysis）或关联规则学习（association rule learning）。隐含关系有2种形式：（1）频繁项集（frequent item sets）表示多个项目经常出现在一起；（2）关联规则（association rules）表示两个项目之间可能有促进关系。
2. **Apriori原理**：Apriori = A Priori（一个先验）。对于频繁项集，如果某个项集是频繁的，那么它的所有子集也是频繁的，子集的子集也是频繁的；如果一个项集是非频繁的，那么它的所有超集也是非频繁的。对于关联规则，如果一个规则不满足最小可信度要求，那么任何左部为该规则左部子集的规则也不会满足最小可信度要求（任何右部为该规则右部超集的规则也不会满足最小可信度要求）。
3. **支持度和可信度（置信度）**：数据集内有N条记录，每条记录包含若干个x y z，支持度和可信度的计算公式是

$$Support（支持度）= \frac{N(x)}{N}$$

$$Confidence（可信度）= \frac{N(x,y)}{N(x)}$$

表示对于包含x的所有记录，规则（$x \rightarrow y$）对其中可信度比例的记录都适用

4. **发现频繁项集**：首先生成所有只包含单个元素的项集，通过扫描数据集过滤掉不满足最小支持度的项集。对剩下的项集进行组合，生成包含2个元素的项集，并重新扫描数据库，再次过滤掉不满足最小支持度的项集。迭代上述过程，每轮迭代中项集包含的元素数目k+1，直到没有包含k个元素的频繁项集被生成（k的最大取值为所有单个元素的数目K）。
5. **发现关联规则**：对频繁项目列表每一层的每个项集（大于最小支持度），都找到组成该项集的单个元素（大于最小支持度）。将单个元素作为关联规则的右部，项集减去该元素的剩余集合作为关联规则的左部（不需要确认是否在频繁项集列表中，因为项集是频繁的，它的所有子集也是频繁的，子集的子集也是频繁的），计算可信度，并与最小可信度比较进行过滤。将通过最小可信度过滤的单个元素组合为包含2个元素的集合（不需要确认是否在频繁项集列表中），并作为关联规则的右部，项集减去该集合的剩余集合作为关联规则的左部（不需要确认是否在频繁项集列表中），计算可信度并进行过滤。将包含2个元素的集合再次进行组合（组合为包含3个元素的集合），进行支持度过滤，进行可信度过滤。重复上述过程，直到集合的元素为项集的元素减1。重复上述过程，直到遍历完频繁项集列表。

算法过程

频繁项集生成：生成所有只包含单个元素的项集，扫描数据集将满足最小Support的项集加入频繁项集列表 → 对单个元素的频繁项集进行组合，生成包含2个元素的项集，并重复最小Support判断过程，结果加入频繁项集列表 → 迭代上述过程，直到没有包含k个元素的频繁项集被生成。频繁项集列表第k-1层的每个项集都包含k个元素（k从1开始）。

关联规则生成：对频繁项目列表每一层的每个项集，都找到组成该项集的单个元素 → 将单个元素作为关联规则的右部，项集减去该元素的剩余集合作为关联规则的左部，计算可信度，并与最小可信度比较进行过滤 → 将通过最小可信度过滤的单个元素组合为包含2个元素的集合，并作为关联规则的右部，项集减去该集合的剩余集合作为关联规则的左部，计算可信度并进行过滤 → 将包含2个元素的集合再次进行组合（组合为包含3个元素的集合），计算可信度并进行过滤 → 重复上述过程，直到集合的元素为项集的元素减1 → 重复上述过程，直到遍历完频繁项集列表

算法特点

- 优点：逻辑简单，项集在升级长度的同时，将低频繁项集不断丢弃
- 缺点：使用在大数据集时，计算量大

机器学习实战

1. Apriori算法
2. 频繁项集生成
3. 关联规则生成
4. 投票中的关联规则发现