# AiLearning之机器学习基础总结

## Logsitic回归：

### sigmoid阶跃函数：

### Tanh函数：sigmoid函数变形，且是0均值的：；

### 寻找最优参数的相关理论：

#### 梯度算子总是指向函数值增长最快的方向。即连续可微函数在某点P，沿在p点的梯度方向有最大的方向导数；

#### 梯度上升算法：与梯度下降算法的区别-目标函数是求最大似然函数：；——批处理学习算法

#### 梯度下降算法：目标函数为求最小损失函数：；

### 梯度上升算法改进：由于梯度上升算法每次更新回归系数时需要遍历整个数据集，当数据集较大时计算成本太高；

#### 随机梯度上升算法：即每次仅用一个样本点来更新回归系数；——在线学习算法；

## Logistic回归和最大熵模型：

### 1. 他们都属于对数线性模型。当类标签只有两个的时候，最大熵模型就是logistic回归模型；

## 多分类标签：逻辑回归也可用于多分类标签。

### 1. 对多分类标签，如果标签A中有a0,a1,a2,....,an个标签，则对每个标签ai，需要训练一个逻辑回归分类器；

### 2. 训练标签ai的逻辑回归分类器时，将标签ai看成一类，非ai标签看成另一类，这样就可以运用二值类逻辑回归分类算法；

### 3.测试数据的时候，将查询点套用在每个逻辑回归分类器中的Sigmoid 函数，取值最高的对应标签为查询点的标签。

## 支持向量机——监督学习算法

### 支持向量：就是离分割超平面最近的那些点；

### 机表示一种算法

## 神经网络 -DNN，CNN

### 神经网络非线性能力及原理：

#### 感知器和逻辑门——》空间分线性切分能力

### BP算法：也叫算法；

### SGD算法：梯度下降算法；

### 回归类问题选择损失函数：MSE-最小平方误差；

### 学习率：

### 常用的package工具： tersenflow， pytorch， keras

## 集成学习

### bagging：“投票选举”——基于数据随机重抽样分类器构造的方法；

#### bagging代表——随机森林算法；

#### 随机森林算法：

####

### boosting：“再学习”——基于所有分类器的加权求和方法；

#### boosting代表——adaboosting；

### bagging和boosting的区别:

#### 1. Bagging与boosting是一种很类似的技术，所使用的多个分类器的类型都是一致的。

#### 2. Bagging是由不同的分类器（1. 数据随机化；2.特征随机化），经过训练，综合得出的出现最多分类结果；

#### 2. Boosting是通过调整已有分类器那些错分的那些数据来获取新的分类器，得出目前最优的结果；

#### 3. Bagging中分类器的权重是相等的；而boosting中的分类器加权求和，所以权重并不相等，每个权重代表的是其分类器在上一轮迭代中的成功度。

## 样例非均衡现象

在分类器训练时，正例数目和反例数目不相等（相差很大）。或者发生在正负例分类错误的成本不同的时候。

###

## 能否使用弱分类器和多个实例来构建一个强分类器？

Adaptive Boosting——自适应boosting

### AdaBoost开发流程：

1. 搜集数据——可以使用任意方法；
2. 准备数据——依赖所使用的弱分类器类型。针对机器学习实战第7章-基于adaboost的分类一节，使用的是单层决策树，该分类器可以处理任何数据类型；
   1. 传统决策树：
      1. 它的层数是一般是由数据的特征数目决定的，每一层的的构建难度主要取决于如何找到这样一个特征属性，使得使用该属性划分当前样本集具有最大信息增益；
      2. 按照上述规则递归构建决策子树，直至某一层子树对应的样本集全部为同一种标签或者样本集里只剩下一种特征属性；
   2. 单层决策树：顾名思义，该分类器只有一层分类节点，该分类节点主要包括划分属性，阈值，判断条件；找到该分类节点的策略就是遍历样本数据的属性，在他们的有限离散值里面找到具有最低分类误差率的阈值；
3. 分析数据：可以使用任意方法；
4. 训练算法：
   1. AdaBoost的大部分时间都用在训练上，分类器将多次在同一数据集上训练弱分类器， 只是每一轮数据的权重不一样:
      1. ；
      2. ；
   2. 每个弱分类器都对应一个单层决策树——：划分属性， 阈值， 判断条件（大于或小于阈值）；
   3. 对每一个弱分类器，通过计算它在样本集上的分类误差率，就可以得到它在最终分类器中的权重：
      1. ;
      2. ；
   4. 注意Adaboost虽然是分迭代训练，但是并不是会一致迭代下去；每一轮迭代训练后，Adaboost对已经训练出来的弱分类器进行加权叠加：
      1. ；
      2. 然后计算f(x)在训练集上的分类误差率，如果为0或者达到指定阈值，就终止迭代；
5. 测试算法：计算分类的错误率；
6. 使用算法：同SVM一样，Adaboost预测两个类别中的一个；如果想把它应用到多类别的场景，则可以固定一个类别，然后将其他类别归类为第二种类别，这样就可以得到多个二值类分类器；

### Adaboost优缺点：

1. 泛化（由具体的、个别的扩大为一般的）错误率低，易编码，可以应用在大部分分类器上，无需参数调节；
2. 缺点：对离群点敏感；
3. 使用数据类型：数值型和标称型数据；

## 无偏估计：

1.定义：就是某个公式对采样后的样本进行统计得到某个指标，这个指标会随着样本的不同而浮动；如果多次采样后，对该指标求期望，如果最终结果与样本变量无关，就说用该指标估计没有系统上的偏差，产生的误差是随机因素造成的；

2.特别地，对样本方差，如果按照：

2.1 计算样本方差，对该方差求期望，最终的结果里会引入样本变量，所以使用该公式对样本进行估计会引入样本偏差；

2.2 为消除样本偏差，需要把样本方差公式调整为：，对该方差公式计算期望，可得到；——此时称该方差公式为样本的无偏估计。

### bagging——并行式集成学习方法的代表

#### 自主采样法——使得初始训练集中约有63.2%的数据流入采样集中；

#### 个体学习器之间无强依赖，可以并行生成；

#### 在对预测输出进行结合时，Bagging通常对分类任务使用简单投票法，对回归任务使用简单平均法；

### AdaBoosting——串行集成学习方法的代表

####

## 回归

### 1. 线性回归

* 1. 使用最小二乘法得到令loss function最小的W，即：—> ;
  2. 使用线性回归得到的是具有最小均方差的无偏估计，有可能出现欠拟合现象，为了解决这个问题，允许在估计中引入一些偏差；
     1. 局部加权线性回归（Locally Weighted Linear Regression，LWLR）
  3. 如果数据的特征数比样本点还多，即X为非满秩矩阵，此时无解；
     1. 解决办法：
        1. 岭回归；
           1. 岭回归就是在矩阵加上一个从而使得矩阵非奇异，进而能对求逆，即回归系数；
           2. 缩减：岭回归实质上是在估计中加上偏差，从而得到更好的估计。这里通过引入来限制了所有w之和，通过引入该惩罚项，能够减少不重要的参数，该技术在统计学中也叫做“缩减”；
        2. Lasso法：
        3. 前向逐步回归：贪心算法，每一步都尽可能减少误差；
           1. 一开始，所有的权重设置为0；
           2. 然后每一步所做的决策就是对某个权重增加或减少一个很小的值，使用新W计算出误差；
           3. 最后求得具有最小误差的回归系数；

### 树回归

数据集中常有输入数据和目标变量之间呈现非线性关系。对这些复杂的关系进行建模，一种可行的方式是使用树来对预测值分段，包括分段常数和分段线性函数。

分段常数——回归树；

分段线性函数——模型树；

1. 回归树 vs 分类树：
   1. 回归树：CART（Classification And Regression trees）：
      1. CART算法用于构建二元树并处理离散型或连续型数据的切分。使用不同的误差准则，就可以通过CART算法构建回归树和模型树；
   2. 分类树：ID3, C4.5
   3. 构建决策树常用的三种方法：ID3, C4.5, CART；他们划分树的分支的方式：
      1. CART做分类工作时，采用GINI值作为节点分裂的依据；回归时，采用样本的最小方差作为样本分裂的依据；
   4. 树剪枝：
      1. 一棵树如果节点过多，表明该模型可能对数据进行了“过拟合”；
      2. 预剪枝（prepruning）；
      3. 后剪枝（postpruning）: 对拥有同一父节点的一组节点进行合并，如果合并后的熵的增加量小于某一阈值，则将改组节点进行合并；
2. 回归树 VS 模型树：
   1. 使用树建模。除了把叶节点简单设为常数值（回归树）外，还可以把叶节点设为分段线性函数（模型树）；如何得到分段线性函数——线性回归；
   2. 决策树较其他机器学习算法的优势之一在于其结果更易于理解。模型树建立在决策树的构建基础上，把切分数据使用线性模型表示，使得其具有更高的可解释性，同时也具有更高的预测准确度；
   3. 为了找到最佳切分，如何计算误差呢？——树建模的关键
      1. 回归树：本质上类似决策树，需要一个指标来表征通过某个特征划分后，划分后的数据集混乱度要比划分前的数据集混乱度有所降低；——对连续型数据，使用数据目标值的最小方差；
      2. 模型树：虽然基于决策树思想划分数据集，但是切分数据后使用线性模型拟合了数据，因此需要使用线性模型预测值和目标值的方差来表征切分能力；、

### 聚类算法

1. K-Means聚类算法：
   1. K-Means术语：
      1. 簇：
      2. 质心：
      3. SSE：簇中所有点到质心的距离平方和；
   2. 聚类的目标：保持簇数目不变的情况下提高簇的质量；
   3. 判断聚类的性能：SSE——SUM of squared Error；
   4. K-Means聚类算法的缺陷：K-Mans算法可能会陷入局部最小值；
      1. 原因：
         1. K值取得不合适，距离函数不合适，随机选择的质心靠的太近等；
         2. 解决办法：二分K-Means聚类算法:
            1. 第一步：将所有点看成一个簇；
            2. 当簇数目小于K时，对每一个簇：

计算总误差；

在给定得簇上面进行K-Means聚类（k=2）;

计算将该簇一分为二之后得总误差；

* + - * 1. 选择使得误差最小得那个簇进行划分操作；
        2. 上述基于SSE的划分过程不断重复，直到得到用户指定的簇数目为止。
        3. biKmeans算法剖析：

如何选择误差最小的簇？（该簇的初始误差也是最大的，经2均值划分后，会使数据集整体的SSE最小）

计算出未二分前的SSE；

遍历每一组簇，从数据集中通过数组过滤器获取该簇对应的数据集，基于该子集进行2-Means聚类，计算该子集进行2-均值聚类后的SSE + 非该子集的所有SSE；选出使得SSE最小的划分簇，并记录进行2-均值划分后的自己SSE；

如何基于最佳2-均值聚类子集划分结果更新数据集的簇信息？

数据集的簇信息包括两方面？ 簇质心（k x N）列表，数据集中每条数据对应的簇索引及及到对应簇质心的SSE（M x 2）

最佳2-均值聚类划分子集是通过数组过滤器从原数据集中过滤出来的，该子集经过2-均值聚类划分后，每条数据在原始数据集中 的顺序是不会变的，所以只需要新得到的0和1的结果簇编号修改为划分簇和新加簇的编号，通过数组过滤器可完成；

更新原质心列表中的第i个质心为二分2-均值聚类后的第一个质心（簇编号为0），然后append二分2-均值聚类后的第二个质心（簇编号为1）了;

### Volitile 原理和使用总结：

1. Volitile作用？——在多线程并发场景中保障共享变量得可见性；
   1. 数据得可见性？——现代操作系统，线程每次读取和写入都不会直接操作主内存，而是每条线程都拥有自己的工作内存，类似缓存，线程在工作内存中保存主内存共享变量的副本，当操作完工作内存的变量后，会写入主内存。多核多线程时代，缓存虽然提高了CPU的执行效率，但是却出现了多线程之间的缓存一致性问题；
      1. 解决办法： 缓存一致性协议；
   2. 编译器重排序、处理器重排序；
   3. Volatile规则：——写操作先于后续的读取操作，保证数据的可见性；
      1. 原理：假如一个共享变量被volatile修饰，该指令在多核处理器下会引发两件事情：
         1. 将当前处理器缓存行数据写回主内存中；
         2. 这个写入的操作会让其他处理器中已经缓存了该变量的内存地址失效，当其他处理器需求再次使用该变量时，必须从主内存中重新读取该值；
         3. 为了保证每个CPU数据的一致性，每一个CPU会通过嗅探总线上传播的数据来检查自己数据的有效性，当发现自己缓存的数据的内存地址被修改，就会让自己缓存该数据的缓存行失效，重新获取数据，从而保证了数据的可见性；
   4. Volatile底层原理：内存屏障；
      1. 即volatile修饰的共享变量在机器指令层面会出现lock 前缀的指令；
      2. 简单来说，被volatile修饰的共享变量，在lock指令后是一个原子操作，而Lock前缀指令就相当于一个内存屏障；
      3. 什么是内存屏障？
         1. 为了提高程序的运行效率，编译器和处理器运行时会对指令进行重排序；
         2. 内存屏障就是一组CPU指令，告诉编译器和CPU，禁止任何指令和当前指令重排序；

### 使用Apriori算法进行关联分析

1. Apriori算法：
   1. 何为关联分析（关联规则学习）？——从大规模数据集中寻找物品间的隐含关系；
   2. 关联分析的难点？如何在合理的时间内找到所有频繁项集；——方法：Apriori算法；
   3. 频繁项集：指经常出现在一块的物品的集合；
      1. 如何度量频繁？——项集的支持度，即数据集中包含该项集的记录所占的比例
   4. 关联规则：描述了两种物品之间可能存在的强关系；
      1. 如何度量关联规则？——可信度或置信度：即支持度({尿布，葡萄酒}) / 支持度({尿布});
      2. 通过暴力遍历所有组合的清单，然后统计每一种组合的频繁程度，太耗时，建议使用Apriori算法；
   5. Apriori算法：
      1. 如果一个项集是非频繁项集，那么它的所有超集也是非频繁的·；
      2. 反过来说，如果某个项集是频繁的，那么它的所有子集也是频繁的；
   6. 从频繁项集中挖掘关联规则：
      1. 频繁项集的量化定义——即该项集需要满足最小支持度要求；
      2. 关联规则的量化定义——可信度：
         1. 一条规则P —> H的可信度定义为support(P | H) / support(P)。
            1. Apriori算法案例：假设规则[0,1,2] ->[3]并不满足最小可信度要求，那么就知道任何左部为[0,1,2]子集得规则也不会满足最下可信度要求；
            2. Q: 上述案例成立的条件是：support([01,2,]) < support([0,1]) ；
            3. 是否可以理解[0,1,2]其实是[0], [1], [2]的数学交集？
         2. Apriori原理另一种表述：如果一个元素项是不频繁的，那么那些包含该元素的超集也是不频繁的；
            1. 通过超集是子集的数学交集可以推断除上述结论；
   7. Apriori算法面对大数据集时高效吗？
      1. 因为每次增加频繁项集的大小，Apriori算法都会重新扫描整个数据集，当数据集很大时，会显著降低频繁项集发现的速度；
      2. 改进：FP-Growth算法，只需要对数据集进行两次遍历，能够显著加快发现频繁项集的速度；
2. FP-Growth算法
   1. 定义：
      1. FP-Growth算法将数据存储在一种称为FP树的紧凑数据结构中。FP代表频发模式（Frequent Pattern）；
   2. 特点：
      1. 一个元素可以在FP树中出现多次，每个树节点上给出集合中单个元素及其在序列中出现的次数；
      2. 另外，FP树会存储项集的出现频率，每个项集会以路径的方式存储在树中。存在相似元素的集合共享树的一部分；
      3. FP树通过链接（link）来连接相似元素，用于快速发现相似项的位置；
   3. 使用FP树发现频繁项集的基本过程：
      1. 构建FP树；
      2. 从FP树中挖掘频繁项集；
         1. 与Apriori算法思路类似，首先从单元素集合开始，然后在此基础上逐步构建更大的集合；
         2. 从FP树中获得条件模式基；
         3. 利用条件模式集，构建一个条件FP树；
         4. 迭代重复上面两个步骤，直到树包含一个元素项为止；
      3. 何为条件模式基（conditional pattern base）？
         1. 对于每一个频繁元素项，它的条件模式基是指以该元素为结尾的路径集合；
      4. 前缀路径：即介于所查找元素项跟树根节点之间的所有内容；
         1. 如何查找一个频繁项的所有前缀路径？
            1. 对树进行穷举式搜索；
            2. 构建FP树时，我们遍历了一次所有项集，构建了包含所有单个元素的头指针表，头指针表包含相同类型元素链表的其实指针；