# 比较全量、抽样与增量三种方式的结果差异

2019 年 10 月 19 日

## 目录



## 1 大规模机器学习的解决方法1——小批量梯度下降法

我们常采用梯度下降法来对算法进行训练，最小化损失函数来优化模型参数。而常用的梯度下降法可以分为三种不同的形式，分别是：

- 批梯度下降法(Batch Gradient Descent, BGD)
- 随机梯度下降法(Stochastic Gradient Descent, SGD)
- 小批量梯度下降法(Mini-batch Gradient Descent, MBGD)

而针对大样本下的机器学习，常用的则是SGD和MBGD，实现类似“增量学习”的效果。这些算法以Linear Model和Deep Learning最为典型。下面介绍一下三种梯度下降的区别。

## 1.1 批梯度下降法

以最简单的线性回归为例，设训练数据集为$D = \{(\boldsymbol{x}^i, y^i)\}_{i=1}^m$，共有$m$个样本，特征维度是$n$，*i.e.* $\boldsymbol{x}^i = (x_1^i, x_2^i, \ldots, x_n^i)^T$，记常数项对应的分量为$x_0^i = 1$，则有$\boldsymbol{x}^i = (x_0^i, x_1^i, x_2^i, \ldots, x_n^i)^T$。线性回归模型的预测函数为

$$h_\theta(\boldsymbol{x}) = \sum_{j=0}^{n} \theta_j x_j$$

对应的Loss function为

$$J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \big(h_\theta(\boldsymbol{x}^i) - y^i\big)^2$$

梯度下降法首先随机初始化参数$\theta$，然后不断反复更新参数，使得更新后的损失函数值相比更新前的损失函数值减少了，这样一步一步的迭代更新，使得损失函数朝着极小值点前进。每轮迭代，参数的更新方式为：

$$\theta_j := \theta_j - \eta \frac{\partial}{\partial \theta_j} J(\theta), \;\; j = 1, 2, \ldots, n$$

当我们只考虑一个数据点$(\boldsymbol{x}, y)$时，$J$对$\theta_j$的偏导数等于：

$$\begin{aligned}
\frac{\partial}{\partial \theta_j} J(\theta) &= \frac{\partial}{\partial \theta_j} \frac{1}{2} \big(h_\theta(\boldsymbol{x}) - y\big)^2 \\
&= \big(h_\theta(\boldsymbol{x}) - y\big) \cdot \frac{\partial}{\partial \theta_j} \big(h_\theta(\boldsymbol{x}) - y\big) \\
&= \big(h_\theta(\boldsymbol{x}) - y\big) \cdot \frac{\partial}{\partial \theta_j} \left( \sum_{i=0}^{n} \theta_i x_i - y \right) \\
&= \big(h_\theta(\boldsymbol{x}) - y\big) x_j
\end{aligned}$$

对于全量训练样本$D$，损失函数$J$对$\theta_j$的偏导数就等于每一个样本上的偏导数的累加求平均：

$$\frac{\partial}{\partial\theta_j}J(\theta)=\frac{1}{m}\sum_{i=1}^{m}\big(h_\theta(\boldsymbol{x^i})-y^i\big)x_j^i$$

因此，批梯度下降法的更新流程如下：

repeat:
**for** $j=0,1,\ldots,n$ **do**
    $\theta_j=\theta_j-\alpha\sum_{i=1}^{m}\big(h_\theta(\boldsymbol{x^i})-y^i\big)x_j^i$
**end for**

从上面的伪代码可以看到，**BGD在更新一次参数$\theta$需要遍历全部的样本(有$m$个样本就要计算$m$个样本对应的梯度)。优点是每次利用全量样本进行更新，参数的优化方向比较稳定，需要的迭代次数比较少。缺点是当样本量很大的时候，BGD计算非常耗时。**因为求最优参数需要多次进行迭代更新，假如更新10轮，就需要遍历整个训练样本10次，计算量太大了。

## 1.2 随机梯度下降法

针对BGD在大样本下的缺点，科学家们提出了随机梯度下降法。这种方法是利用每个样本损失函数的偏导数来更新$\theta$，其更新流程如下：

Randomly shuffle training samples
repeat:
**for** $i=1,2,\ldots,m$ **do**
    **for** $j=0,1,\ldots,n$ **do**
        $\theta_j=\theta_j-\eta\big(h_\theta(\boldsymbol{x^i})-y^i\big)x_j^i$
    **end for**
**end for**

从上面的伪代码看出，SGD是每来一个样本，就计算梯度并更新一次参数，对比批梯度下降法，SGD的优点是计算速度快，这恰好弥补了BGD的缺点。但是SGD也有一个问题就是会有噪音的问题。**因为它每次只用一个样本来更新参数，所以其在解空间中的搜索过程看起来会比较飘忽，但是随着迭代次数的增加，总体上还是朝着最优值的方向移动的。**

### 1.3 小批量梯度下降法

BGD和SGD各有各的优缺点，而这两种算法的折中结果就是Mini-batch Gradient Descent，MBGD每次不仅仅用一条样本，而是用一小部分的数据来进行训练，更新参数。**相比于BGD，它训练速度更快；而相比于SGD，它最终求得的参数的准确率也更高（或者说稳定性更好）**。下面是MBGD的更新流程：

1: m=1000, b=10 ▷ 样本量为1000，批大小为10
2: repeat:
3: **for** $i = 1, 11, 21 \ldots, 991$ **do**
4: **for** $j = 0, 1, \ldots, n$ **do**
5: $\theta_j = \theta_j - \alpha \sum_{k=i}^{i+b} \left(h_\theta(\boldsymbol{x^k}) - y^k\right) x_j^k$
6: **end for**
7: **end for**

因此，对于大样本量的模型训练，可以用mini-batch的方法，批量进行训练更新参数。当然只迭代一轮是不够的，通常需要迭代多轮才能得到较为准确的结果。

## 2 大规模机器学习的解决方法2——增量学习

本节内容主要参考论文A Survey on Incremental Learning

### 2.1 增量学习的背景

对数据进行自动分类的一个前提假设是基于统计学习方法的，即用于训练分类模型的训练数据和待分类的测试数据来自相同的总体分布，并且两者的特征空间是相同的。然而随着时间的流逝，数据的分布和数据的特征可能会发生变化，因此基于历史数据所训练的模型可能将不再适用于某些新的数据。所以对于一个分类系统来说，可以实现增量的学习以应对不断变化的数据进行正确的分类是很有必要的。这就是增量学习的概念。

而具体的，数据的分布和特征变化可以分成下面三类：

- 样本的变化：即样本特征值的变化，或者是样本各个类别的比例变化
- 类别的变化：新数据中可能会出现新的类别，导致原来的分类模型需要改变。

- 特征的变化：训练数据和测试数据的特征空间不一致

而为了应对数据可能产生的这些变化，出现了增量学习(Incremental Learning)。类似的，也可以分成三个类别：

- 样本增量学习(SIL)：**在保有现有知识的情况下，通过新样本的增量学习，提取新的知识，融合旧的知识来提高分类的准确性。**
- 类别增量学习(CIL)：识别新类，并将其加入现有的类别集合中，提升分类准确性
- 特征增量学习(FIL)：一些新的特征可以提升分类准确率，在现有特征空间的基础上，构建新的特征空间

## 2.2 增量学习的特点

增量学习主要有下面四个特点：

1. 分类器可以从新的数据中学得知识
2. 学习和训练的数据只是新数据。不需要重复训练之前的训练数据。举个例子，假设有200条数据，用增量训练的方法，第一次训练100条，第二次训练100条，则在第二次训练的时候，前100条数据已经不存在了，不过他们包含的信息存在于现有的模型中。
3. 模型在学习新的样本时，是建立在之前阶段已有的模型基础上的，不仅复习了旧有的知识，同时整合了新学到的知识，最后返回一个更复杂更准确的模型。
4. 可以识别并分类新的类别

## 2.3 增量学习和批量训练的区别

第一节说到的利用Mini-batch Gradient Descent来更新模型参数的方法，以及这里的样本增量学习方法，都可以解决数据量太大，无法读入内存进行训练的问题。这两者的区别在于：

- batch训练：越往后的batch对模型的影响越大。假如后面输入的数据质量比较差（比如分布不均匀，或者有很多异常值），则模型会去努

力的拟合这部分数据，导致最终结果会被这部分数据带偏。如果要用batch训练的话，尽量保证增量数据的质量。

- 增量学习：增量学习在每次更新模型的时候，会综合考虑原有的模型和新数据之间的tradeoff。通常，**使用增量训练的分类算法，其本身的分类原则就是自然支持增量学习的。**具体地，比如朴素贝叶斯、支持向量机、决策树、KNN等分类算法都是自然支持增量学习的。

在python中，无论是sklearn库里的`partial_fit()`，还是xgboost/lightgbm的增量训练方法，**我们使用这些方法来进行增量训练的前提就是这些机器学习算法本身就具有增量学习的特性。而类似Mini-batch Gradient Descent的训练方法，更多的属于一种训练的trick。**

# 3 实验

原始的总数据集很大有接近1800万行，而这里的实验用之前抽样的数据作为假想中的“全量数据”，即：总样本量为89万行，用户个数为5436个。（把这个数据集看成全量的总体数据集，然后分析抽样结果与增量训练的结果对比）实验中的抽样数据，则是随机抽取了这个假想的“全量数据集”的用户，取5%的用户，这些用户的行为数据构成了抽样训练集，大约有5万行。

Xgboost提供的增量训练方法，是从一个现有的树出发，只更新现有的树结构，基于新一批的数据，更新原有树节点的统计量和叶子节点的值。模型的参数较之前没有调整，每轮迭代读入3万/5万行数据进行incremental training，具体设置的api参数如下：

- `process_type = update`: 从已有的模型出发，保留树的结构不变
- `updater = refresh`: 指定每轮迭代时树的更新方式。`refresh`表示利用新的数据，刷新原有树的内部节点的统计量。这里不会进行随机行采样。
- `refresh_leaf = True`: 关于`updater = refresh`的一个参数，设置为`True`时，不仅更新树的内部节点统计量，还会刷新叶子节点的输出值。

### 3.1 按默认数据集顺序读入

默认的训练集是按照动漫的id排序的，即不同用户不同时间更新的关于同一部动漫的评分以及观看状态，在训练集中是处在连续的一个片段中。结果如下：

表 1: Table 3.1 Incremental training with default order

| 召回组合 | 召回率 | 精准率 | 命中率 | 覆盖率(占比) | 覆盖率(信息熵) | 覆盖率(type的信息熵) |
|---|---|---|---|---|---|---|
| 90w“全量”数据 | 7.469% | 11.203% | 77.872% | 21.296% | 8.067632605 | 1.333672452 |
| 5%用户抽样(5.1w) | 6.242% | 8.889% | 78.571% | 9.148% | 8.148716012 | 1.390282018 |
| 增量训练(5w)首轮 | 6.819% | 10.228% | 75.574% | 22.256% | 8.151649871 | 1.351169404 |
| 增量训练(5w)末轮 | 7.127% | 10.690% | 76.824% | 18.056% | 7.936399226 | 1.303683283 |
| 增量训练(3w)首轮 | 6.561% | 9.841% | 74.325% | 23.680% | 8.208354503 | 1.364881415 |
| 增量训练(3w)末轮 | 7.099% | 10.648% | 76.743% | 17.397% | 7.931140783 | 1.297827393 |

从上表可以看出：从首轮到末轮，模型的预测效果有了较为明显的提高，更接近于全量数据下的表现，同时显著优于抽样数据下的表现。

### 3.2 按时间顺序读入

上面的默认数据集顺序导致了每次读取的part of data只包含小部分动漫的信息。这一次把全量数据集按照用户更新状态打分的时间排序，然后按时间顺序由远及近读入。结果如下：

表 2: Table 3.2 Incremental training with update date order

| 召回组合 | 召回率 | 精准率 | 命中率 | 覆盖率(占比) | 覆盖率(信息熵) | 覆盖率(type的信息熵) |
|---|---|---|---|---|---|---|
| 90w“全量”数据 | 7.469% | 11.203% | 77.872% | 21.296% | 8.067632605 | 1.333672452 |
| 5%用户抽样(5.1w) | 6.651% | 9.471% | 79.365% | 7.469% | 7.825825802 | 1.321671186 |
| 增量训练(5w)首轮 | 7.203% | 10.804% | 76.179% | 22.166% | 8.226319 | 1.330604 |
| 增量训练(5w)末轮 | 7.271% | 10.906% | 77.630% | 17.622% | 7.812633 | 1.300618 |
| 增量训练(3w)首轮 | 7.210% | 10.815% | 76.421% | 21.191% | 8.160203 | 1.332818 |
| 增量训练(3w)末轮 | 7.185% | 10.777% | 78.033% | 17.532% | 7.780351 | 1.309416 |

从上表可以看出：从首轮到末轮，模型的预测效果基本没有什么变化，从首轮开始其实就比较接近于全量数据下的表现了，每轮的更新迭代也只是在上下震荡。效果仍然显著优于抽样的结果

### 3.3 按随机顺序读入

最后，试着把全量数据集随机shuffle打乱，然后依次读入，没有任何顺

序。结果如下：

表 3: Table 3.3 Incremental training with random order

| 召回组合 | 召回率 | 精准率 | 命中率 | 覆盖率(占比) | 覆盖率(信息熵) | 覆盖率(type的信息熵) |
|---|---|---|---|---|---|---|
| 90w“全量”数据 | 7.469% | 11.203% | 77.872% | 21.296% | 8.067632605 | 1.333672452 |
| 5%用户抽样(5.1w) | 6.651% | 9.471% | 79.365% | 7.469% | 7.825825802 | 1.321671186 |
| 增量训练(5w)首轮 | 7.106% | 10.658% | 76.542% | 22.091% | 8.07655 | 1.372281 |
| 增量训练(5w)末轮 | 7.323% | 10.984% | 77.227% | 17.697% | 7.831545 | 1.315776 |
| 增量训练(3w)首轮 | 7.250% | 10.874% | 77.106% | 22.091% | 8.042624 | 1.354189 |
| 增量训练(3w)末轮 | 7.239% | 10.858% | 77.429% | 17.082% | 7.809792 | 1.308236 |

从上表可以看出：从首轮到末轮，模型的预测效果有些微提升，从首轮开始比较接近于全量数据下的表现了，后续的更新在此基础上略有提升。同样的，也是显著优于抽样结果。

### 3.4 小结

- 从上面三个实验可以看出，xgboost的增量训练还是起到了一定的效果的
- 根据不同的数据读入顺序，增量训练的结果也会有所改变
- 基本上其模型效果都优于单纯的对用户采样的结果，并略逊于全量数据的结果
- batch=5w的效果要略优于batch=3w的效果

## 4 随机抽样

上面三个实验中，抽样数据的结果和增量的结果比起来差了挺多的，下面对全量训练集（90w）的用户，重复多抽几次样（固定每次5%，更换随机数种子），看看总体的表现。

**Table 3.4** Random sampling for users

| 5%用户采样seed | 召回率 | 精准率 | 命中率 | 覆盖率(占比) | 覆盖率(信息熵) | 覆盖率(type的信息熵) |
|---|---|---|---|---|---|---|
| 1001 | 6.242% | 8.889% | 78.571% | 9.148% | 8.148716 | 1.390282 |
| 1002 | 6.603% | 11.093% | 77.600% | 10.063% | 8.253565 | 1.476714 |
| 1003 | 6.985% | 10.876% | 83.898% | 9.223% | 8.237271 | 1.374655 |
| 1004 | 7.422% | 10.509% | 75.424% | 9.553% | 8.205792 | 1.439307 |
| 1005 | 6.274% | 9.546% | 68.182% | 9.373% | 8.101647 | 1.342534 |
| Average | 6.705% | 10.182% | 76.735% | 9.472% | 8.1893982 | 1.4046984 |

上表可以看出，随机抽样的结果彼此之间也有着一定的差异，不过平均来说其效果还是低于增量训练以及全量结果的。而上次对“真实全量数据”用增量训练预测的方法得到的结果，对比随机抽样得到的结果（90w数据，5400用户)，二者非常接近。这是不是可以说明，**因为样本量的关系，90w的样本量得到的结果已经很不错了。（作为对比，如果全量是90w的话，抽样得到的5w数据的结果就显得比较单薄）**

关于xgboost的增量训练方法，官方文档只有一点点说明，没有查到相关的具体实现方法，不知道具体是如何用新的batch样本来更新现有树的叶子输出以及内部节点阈值。