Boost介绍

Jianjun Wu

September 3, 2018

1 回顾

决策树是最广为人知的数据挖掘算法之一。其中:

- ID3和C4.5用于构建多叉分类树，要求属性全都是离散取值。每次分裂时选择信息增益最大的特征A作为分裂特征，设A有k个可能的取值, 然后根据样本在A 上的取值是这k个中的哪一个进而将样本划分到k个子节点中的一个。

- CART 构建二叉树，既可以用于分类，也可以用于回归。用于回归时，假定属性全都是连续取值，在每个属性的每种取值上做二分(大于还是小于)，取其中导致的回归损失最小者作为划分点。用于分类时，要求属性全都离散取值，计算以每个属性的每个离散取值点做划分(是否等于)时Gini指数最小者做为划分点。

可用于决策树的集成学习方法有：

- Bagging。对样本集进行多次有放回的采样从而构建多个样本集，每个样本集训练一颗决策树，最后多颗决策树基于投票或者平均的方式输出集成预测值。

- 随机森林(Random Forests)。首先对训练集做行采样选择一批子样本，然后做列采样选择一个特征子集，接着根据得到的训练集建立决策树，重复这个过程便得到多颗树。

- Boost。前两种方法各颗树是并行且独立地建立，而Boosting方法中的各颗树是依次建立且后面树的会利用前面那些树的信息。

2 Boost技术

AdaBoost是非常著名的Boost算法，他的思想就是顺序地建立一颗颗的分类树，每次建立树之前，先增大那些被错误分类样本的权重，使得后续的分类树关注于那些难分的样本。

AdaBoost是**前向分步算法**1的特例，这个结论的详细证明可以参考李航老师的<< 统计学习方法 >> 一书。

这里需要特别指出的有几点。

- 首先，我们的模型是加法模型(式1)，最终的模型是一个个基本模型线性相加得到的，从而我们需要求解的问题为:

$$min \sum_{i=1}^{n} L(y_i, \sum_{m=1}^{M} \beta_m b(\mathbf{x}_i; \gamma_m))$$

Algorithm 1 前向分步算法

Require:

训练数据集$T = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$.

基本模型对应的函数集$\{b(\mathbf{x}; \gamma)\}$

损失函数$L(y, f(\mathbf{x}))$

Ensure:

加法模型:$f(\mathbf{x}) = \sum_{m=1}^{M} \beta_m b(\mathbf{x}; \gamma_m)$

1: 初始化$f_0(\mathbf{x})$;

2: **for** each $m \in [1, M]$ **do**

3: 求解下面的问题:

$$(\beta_m, \gamma_m) = argmin_{\beta,\gamma} \sum_{i=1}^{n} L(y_i, f_{m-1}(\mathbf{x}_i) + \beta b(\mathbf{x}_i; \gamma))$$

4: 更新:

$$f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}_i) + \beta_m b(\mathbf{x}_i; \gamma_m)$$

5: **end for**

6: **return**

$$f(\mathbf{x}) = \sum_{m=1}^{M} \beta_m b(\mathbf{x}_i; \gamma_m) \tag{1}$$

- 另外，我们求解的思想是前向分步的，也就是在已有模型上，一个一个地求解后续最优的基本模型。
- 最后，不同的损失函数可以导出不同的算法，比如损失函数为指数损失函数时就可以导出AdaBoost算法。

可以认为，上面三点是Boost算法最关键的特征。

下面，我们再来解读大名鼎鼎的GBDT(Gradient Boosting Decision Tree)算法2。可以看出，GBDT是一种Boost算法，因为他有Boost 算法的三个特征，所以我们很轻易的理解字母B，至于字母DT，我们只需要把前向分步算法中的基本模型换成决策树就可以了，那么G怎么理解呢?在本文作者看来，G的理解不是很容易。尽管我们已被告知GBDT算法的流程和G是什么怎么算，但是我们需要搞清楚为什要G?

最常见的机器学习问题可以形式化为: 给定训练数据$(\mathbf{x}_i, y_i), i = 1, 2, \ldots, n$，根据我们的目标选择一个合适的损失函数$L(y, \hat{y})$, 选定我们的模型函数族$\hat{y} = f(\mathbf{x}, \theta)$，接着加上一个对模型函数中参数做正则化的项$\Omega(\theta)$，然后

Algorithm 2 GBDT算法

Require:

训练数据集$T = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$.

损失函数$L(y, f(\mathbf{x}))$

1: 初始化$f_0(\mathbf{x}) = argmin_c \sum_{i=1}^{n} L(y_i, c)$;

2: **for** each $m \in [1, M]$ **do**

3: 对每个样本i计算:

$$r_{mi} = -\left[\frac{\partial L(y_i, f(\mathbf{x}_i))}{\partial f(\mathbf{x}_i)}\right]\Bigg|_{f(\mathbf{x}_i)=f_{m-1}(\mathbf{x}_i)}$$

4: 计算一颗最优回归树$b(\mathbf{x})$(注意此时回归的损失函数是$L(y, f(\mathbf{x}))$)，拟合数据集$\{(\mathbf{x}_1, r_{m1}), (\mathbf{x}_2, r_{m2}), \ldots, (\mathbf{x}_n, r_{mn})\}$.

5: 更新

$$f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + b(\mathbf{x})$$

6: **end for**

7: **return** $f_M(\mathbf{x})$;

求解下面的问题:

$$minarg_\theta \sum_{i=1}^{n} L(y_i, f(\mathbf{x}_i, \theta)) + \Omega(\theta) \tag{2}$$

为了求解这个问题，我们会求助于最优化的工具，常常我们会利用类似梯度下降的方法迭代求解这个问题。大多数场景下，模型参数θ 是处于连续的实数空间上的，比如SVM 模型中的参数是系数，神经网络中参数是神经元之间的连接权重等，他们都是连续实数，所以梯度下降等优化策略可用。然而，一旦模型参数不连续，甚至不是实数，那么该怎么办呢？比如，决策树模型，他的参数其实是整颗树的结构，我们很难用一个实数函数来表示他，此时，我们就要考虑在非实数空间上如何优化模型了。而GBDT就是这样一个算法，他在决策树结构这个参数空间上求解问题2，再次强调一下，此时模型的参数θ是树结构，而不是实数。

现在，我们就来以非实数空间上的优化这个角度来解释GBDT。我们希望对训练集$(\mathbf{x}_i, y_i), i = 1, 2, \ldots, n$，计算$\hat{y}_i, i = 1, 2, \ldots, n$，使得下面的损失最小:

$$min_{\{\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_n\}} \sum_{i=1}^{n} L(y_i, \hat{y}_i) \tag{3}$$

注意此时我们问题的变量为$\hat{y}_i$，他仍是一个实数，那么我们可以采用梯度下降不断调整$\hat{y}_i$使得损失最小，也就是

$$\hat{y}_i^{m+1} = \hat{y}_i^m - \eta \frac{\partial L(y_i, \hat{y}_i)}{\partial \hat{y}_i}\bigg|_{\hat{y}_i=\hat{y}_i^m} \tag{4}$$

m表示迭代轮数。当然，问题(3)的最佳解为$\hat{y}_i = y_i$。然而，我们其实是想通过一个模型计算得到每个$\hat{y}_i$，也就是$\hat{y}_i = F(\mathbf{x}_i;\theta)$，然后希望算出来的这个$\hat{y}_i$最小化式(3)，所以至于能否使得$\hat{y}_i = y_i$成立取决于我们的模型。在常见的机器学习方法学中，我们通过优化算法迭代地调整模型参数θ，使得最后模型计算得到的$\hat{y}_i$ 最小化损失，然而优化算法往往要求$\theta \in \mathbf{R}^n$，也即是参数是处于实数空间的。现在我们采用一种不同的方法学，这种方法可以不限定参数处于实数空间。我们通过式(4) 迭代地求解$\hat{y}_i$，然后再要求模型通过调整自己努力拟合最新的$\hat{y}_i$,，最后得到的模型便可以输出较小损失的$\hat{y}_i$。进一步，为了简化问题，我们限定模型的结构为一个加法模型:

$$F(\mathbf{x};\theta) = \sum_{m=0}^{M} f_m(\mathbf{x})$$

注意，根据梯度下降式(4):

$$\hat{y}_i^M = \hat{y}_i^0 - \sum_{m=1}^{M} \eta^m \frac{\partial L(y_i, \hat{y}_i)}{\partial \hat{y}_i}|_{\hat{y}_i^m} = \hat{y}_i^0 - \sum_{m=1}^{M} r_{im} \tag{5}$$

这里令$\eta^m \frac{\partial L(y_i,\hat{y}_i)}{\partial \hat{y}_i}|_{\hat{y}_i^m} = r_{im}$。至此，可以发现,如果令$f_0(\mathbf{x}_i) = \hat{y}_i^0$,然后每次迭代$m$中新增一个$f_m(\mathbf{x})$个使得: $f_m(\mathbf{x}_i) = r_{im}$，那么我们其实就是在执行式(4)所定义的梯度下降过程。

3 XGBoost原理

XGBoost [1]是一款实现了改进过的Boost算法的软件，而这个改进过的Boost算法在给定训练数据$\{(\mathbf{x}_i, y_i)\}_{i=1}^n$后，求解下面的问题：

$$min \sum_{i=1}^{n} L(y_i, \widehat{y_i}) + \sum_{k=1}^{K} \Omega(f^k) \tag{6}$$

其中，$\widehat{y_i} = \sum_{k=1}^{K} f^k(\mathbf{x}_i)$，$f^k$ 表示一颗回归树，它有T^k个叶子节点，每个叶子节点的回归值为$\mathbf{w}_i^k, i = 1, 2, \ldots, T^k$，而$\Omega(f^k) = \gamma T^k + \frac{1}{2}\lambda \|\mathbf{w}^k\|^2$表示正

则项。可以看出，我们采用的仍然是加法模型，同样地，我们采用前向分步的方法求解。也就是每一个前向步t中，我们求解下面的问题:

$$min_{f^t}\mathcal{L}^t = \sum_{i=1}^{n} L(y_i, \widehat{y}_i^{(t-1)} + f^t(\mathbf{x}_i)) + \Omega(f^t) \tag{7}$$

也就是，在每一个前向步中，我们需要选择合适的f^t使得$\mathcal{L}^t$最小。对L在点$\widehat{y}_i^{(t-1)}$做taylor二阶展开则有:

$$L(y_i, \widehat{y}_i^{(t-1)} + f^t(\mathbf{x}_i)) \approx L(y_i, \widehat{y}_i^{(t-1)}) + g_i^{t-1} f^t(\mathbf{x}_i) + \frac{1}{2} h_i^{t-1} (f^t(\mathbf{x}_i))^2 \tag{8}$$

其中，

$$g_i^{t-1} = \left.\frac{\partial L(y_i, y)}{\partial y}\right|_{y=\widehat{y}_i^{(t-1)}}$$

$$h_i^{t-1} = \left.\frac{\partial^2 L(y_i, y)}{\partial^2 y}\right|_{y=\widehat{y}_i^{(t-1)}}$$

式(7)可以近似为：

$$min_{f^t}\mathcal{L}^t \approx \widetilde{\mathcal{L}}^t = \sum_{i=1}^{n} [L(y_i, \widehat{y}_i^{(t-1)}) + g_i^{t-1} f^t(\mathbf{x}_i) + \frac{1}{2} h_i^{t-1} (f^t(\mathbf{x}_i))^2] + \Omega(f^t) \tag{9}$$

去掉f^t的常数项有:

$$\widetilde{\mathcal{L}}^t = \sum_{i=1}^{n} [g_i^{t-1} f^t(\mathbf{x}_i) + \frac{1}{2} h_i^{t-1} (f^t(\mathbf{x}_i))^2] + \Omega(f^t) \tag{10}$$

现在，我们在每一个前向步中，需要选择合适的回归树f^t使得式(10)最小。我们定义I_j^t为落在回归树f^t的叶节点j中的训练样本集，且把Ω的定义带入，从而式(10)可以改写为:

$$\begin{aligned}
\widetilde{\mathcal{L}}^t &= \sum_{j}^{T^t} \sum_{i \in I_j^t} [g_i^{t-1} f^t(\mathbf{x}_i) + \frac{1}{2} h_i^{t-1} (f^t(\mathbf{x}_i))^2] + \gamma T^t + \frac{1}{2} \lambda \|\mathbf{w}^t\|^2 \\
&= \sum_{j}^{T^t} \sum_{i \in I_j^t} [g_i^{t-1} w_j^t + \frac{1}{2} h_i^{t-1} (w_j^t)^2] + \gamma T^t + \frac{1}{2} \lambda \sum_{j}^{T^t} (w_j^t)^2 \\
&= \sum_{j}^{T^t} [(\sum_{i \in I_j^t} g_i^{t-1}) w_j^t + \frac{1}{2} (\sum_{i \in I_j^t} h_i^{t-1} + \lambda)(w_j^t)^2] + \gamma T^t
\end{aligned} \tag{11}$$

令$G_j^t = \sum_{i \in I_j^t} g_i^{t-1}$, $H_j^t = \sum_{i \in I_j^t} h_i^{t-1}$ 那么式(11)可以写为:

$$\widetilde{\mathcal{L}}^t = \sum_j^{T^t} [G_j^t w_j^t + \frac{1}{2}(H_j^t + \lambda)(w_j^t)^2] + \gamma T^t \tag{12}$$

请注意根据前向分步算法，我们要最小化$\mathcal{L}^t$，变量本来是回归树f^t。通过taylor近似后我们最小化$\widetilde{\mathcal{L}}^t$ ，再经过等价变换后我们得到式(12)。此时可以看出回归树f^t 的叶子节点个数T^t，以及每个叶子节点的回归值w_j^t是我们的变量，我们需要调整二者使得式(12)达到最小值。为此，我们采用常见求解回归树结构的策略。也就是先固定树结构，从而T^t也就是固定，优化w_j^t，也就是给每个叶节点计算最优的回归值。然后寻找当前树上最佳的分裂，从而让叶节点个数加一。T^t固定时，$\widetilde{\mathcal{L}}^t$ 是关于w_j^t 的二次函数，可以得到其解析解:

$$w_j^t = \frac{-G_j^t}{H_j^t + \lambda} \tag{13}$$

此时$\widetilde{\mathcal{L}}^t$的最小值为:

$$\widetilde{\mathcal{L}}^t = \sum_j^{T^t} [-\frac{1}{2}\frac{(G_j^t)^2}{H_j^t + \lambda}] + \gamma T^t \tag{14}$$

接下来，我们寻找最佳的分裂。叶节点j做一次二分后得到两个新的叶节点j_L, j_R，此时再求新的两个新叶节点最佳的回归值$w_{j_L}^t, w_{j_R}^t$，此时$\widetilde{\mathcal{L}}^t$的变化值为:

$$split = \frac{1}{2}\frac{(G_{j_L}^t)^2}{H_{j_L}^t + \lambda} + \frac{1}{2}\frac{(G_{j_L}^t)^2}{H_{j_L}^t + \lambda} - \frac{1}{2}\frac{(G_j^t)^2}{H_j^t + \lambda} - \gamma \tag{15}$$

显然，我们希望$split$越大越好。至此，我们得到了XGBoost中树节点分裂的原则。

4 树节点分裂算法

我们这里只考虑树节点的二分，多分暂不考虑。

对于连续属性，通常先排序，然后按顺序依次以每个取值对节点进行二分，左边的为小于等于取值的样本集合，右边是大于取值的样本集合。连续属性分裂一个重要问题是候选分裂点的生成，如果按顺序把每个取值都当做候选分裂点，则计算量会很大，所以常常只把某些分位数的取值作为候选分裂点。

4.1 分位数近似计算

[2] 提出了一种online的分位数近似算法，简称GK算法，GK算法是截止现在支持ϵ近似分位数查询算法中空间复杂度最小的确定性算法(目前属于随机算法的KLL [3]算法可以有更小的空间复杂度)。在时刻n(表示已经观察到n 个数据)，该算法维护一个集合$S(n) = \{t_i | i = 0, 1, \ldots, s-1\}$，该集合被称为summary。其中$t_i = (v_i, g_i, \Delta_i)$，$v_i$表示观测序列中某个数据，并且

$$g_i = r_{min}(v_i) - r_{min}(v_{i-1}), \quad \Delta_i = r_{max}(v_i) - r_{min}(v_i)$$

$r_{min}(v)$ 和$r_{max}(v)$分别表示截止时刻n，v 在已观测到的前n个元素中秩的下界和上界的估计，并且$S(n)$中的元素按照v_i的取值大小升序排列，v_0表示前n个元素的最小值，v_{s-1}示前n个元素的最大值，$s = |S(n)|$。

可以发现:

$$r_{min}(v_i) = \sum_{j=0}^{j \le i} g_j + r_{min}(v_0) = \sum_{j \le i} g_j \tag{16}$$

注意，$r_{min}(v_0) = 0$，从而

$$r_{max}(v_i) - r_{min}(v_{i-1}) = \sum_{j \le i} g_j + \Delta_i - \sum_{j \le i-1} g_j = g_i + \Delta_i \tag{17}$$

所以，在区间(v_{i-1}, v_i)内最多有$g_i + \Delta_i - 1$ 个观测值。

现在证明$S(n)$能支持分位数的近似查询，并且查询误差有上界。

Lemma 1. *查询任意分位数ϕ，$S(n)$总中存在一个数据，其秩与真实分位数的秩之间的误差小于：$max_i \frac{g_i+\Delta_i}{2}$。*

Proof. 设$r = \lceil \phi n \rceil, e = max_i \frac{g_i+\Delta_i}{2}$，我们需要在$S(n)$中找到一个元素$v_i$使得

$$r - e \le r_{min}(v_i) \le r_{max}(v_i) \le r + e$$

如果$r \ge n - e$，我们有$r_{min}(v_{s-1}) = r_{max}(v_{s-1}) = n$($v_{s-1}$确定总是最大值)，并且：

$$r \ge n - e \Rightarrow n \le r + e$$

$$n \ge r(\text{因为} r = \lceil \phi n \rceil) \Rightarrow n > r - e$$

所以v_{s-1}即是需要的数据。

如果$r < n - e$，那么我们首先找到满足$r_{max}(v_j) > r + e$的最小的j(注意，我们总是可以找到这样的j，比如$j = s - 1$，则$r_{max}(v_{s-1}) = n$ ，

且$r < n - e \Rightarrow n > r + e$，所以$r_{max}(v_{s-1}) > r + e)$。找到这样的$j$后，我们可以证明：

$$r - e \leq r_{min}(v_{j-1})$$

，否则$r - e > r_{min}(v_{j-1}) \Rightarrow r > r_{min}(v_{j-1}) + e$，所以$r_{max}(v_j) > r + e \Rightarrow r_{max}(v_j) > r_{min}(v_{j-1}) + e + e$，由式(17)得：

$$r_{max}(v_j) = r_{min}(v_{j-1}) + g_j + \Delta_j > r_{min}(v_{j-1}) + 2e \Rightarrow \frac{g_j + \Delta_j}{2} > e$$

从而与$e = max_i \frac{g_i + \Delta_i}{2}$矛盾。接下来我们证明：

$$r_{max}(v_{j-1}) \leq r + e$$

因为我们规定了

$$j = min(j | r_{max}(v_j) > r + e)$$

也就是j是满足$r_{max}(v_j) > r + e$的最小序号，再小必然不满足这个不等式，所以$r_{max}(v_{j-1}) \leq r + e$。所以$v_{j-1}$正是所需的数据。 □

根据lemma 1，只要保证对于summary $S(n)$在任意时刻n有

$$max_i(g_i + \Delta_i) < 2\epsilon n$$

，那么对于任何分位数查询ϕ，$S(n)$总能返回一个v，其秩$r(v)$ 落在区间$[\lceil \phi n \rceil - n\epsilon, \lceil \phi n \rceil + n\epsilon]$内。接下来我们需要找到方法确保summary满足这个性质。

GK算法的工作流程可以概括为：每出现一个新观测数据v，则向summary中插入一个新元组$t = (v, 1, \lfloor 2\epsilon n \rfloor)$，然后周期性地合并summary中的元组。

元组t_i 在时刻n时的容量定义为：$cap(t_i, n) = \lfloor 2\epsilon n \rfloor - \Delta_i$。GK算法把元组容量的取值空间划分成一个个band，令$p = \lfloor 2\epsilon n \rfloor$，定义对于元组$t_i$，对于$[1, \log p)$中整数$\alpha$，如果

$$2^{\alpha-1} + p \bmod 2^{\alpha-1} \leq cap(t_i, n) < 2^{\alpha} + p \bmod 2^{\alpha}$$

则说t_i在时刻n属于$band_\alpha$，记为$band(t_i, n) = \alpha$。读者可能注意到了，$cap(t_i, n)$会随着时间n变化，但是这样定义band可以使得：如果两个元组t_i, t_j在时刻n属于同一个band，那么在$n'(n' > n)$ 时他们仍然会属于同一个band。注意每次插入的新元组，其容量为0，我们把这些容量为0 的元组的band记为$band_0$。我们定义$band_\alpha(n)$为在时刻n所有band取值为α 的元组集合。

Lemma 2. *$band_\alpha(n)$中的全部元组总共有2^α或$2^{\alpha-1}$个不同的$cap(t_i, n)$值。*

Proof. $band_\alpha(n)$的左边界是$2^{\alpha-1}+p\,mod\,2^{\alpha-1}$，右边界为$2^\alpha+p\,mod\,2^\alpha$，而

$$
\begin{aligned}
&p\,mod\,2^\alpha < 2^{\alpha-1}, 令r = p\,mod\,2^\alpha，则p = m*2^\alpha + r\\
&\Rightarrow p = 2m*2^{\alpha-1}+r \quad ,注意r<2^{\alpha-1}\\
&\Rightarrow p\,mod\,2^{\alpha-1} = r = p\,mod\,2^\alpha
\end{aligned}
$$

，所以有$p\,mod\,2^\alpha < 2^{\alpha-1} \Rightarrow p\,mod\,2^\alpha = p\,mod\,2^{\alpha-1}$，此时右边界减去又边界得$2^\alpha - 2^{\alpha-1} = 2^{\alpha-1}$，此时$band_\alpha(n)$中有$2^{\alpha-1}$个不同的值。又因为：

$$
\begin{aligned}
&p\,mod\,2^\alpha \geq 2^{\alpha-1}, 令r = p\,mod\,2^\alpha, p = m*2^\alpha + r\\
&\Rightarrow 因为r \geq 2^{\alpha-1},且r<2^\alpha,有r = k*2^{\alpha-1}+l,并且k=1,否则r >= 2^\alpha,\\
&\Rightarrow r = 2^{\alpha-1}+l \Rightarrow p = 2m*2^{\alpha-1}+2^{\alpha-1}+l\\
&\Rightarrow p\,mod\,2^{\alpha-1} = l\\
&\Rightarrow 2^{\alpha-1}+p\,mod\,2^{\alpha-1} = 2^{\alpha-1}+l = r = p\,mod\,2^\alpha
\end{aligned}
$$

，所以有$p\,mod\,2^\alpha \geq 2^{\alpha-1} \Rightarrow p\,mod\,2^\alpha = 2^{\alpha-1}+p\,mod\,2^{\alpha-1}$，此时右边界减去又边界得$2^\alpha$,此时$band_\alpha(n)$中有$2^\alpha$个不同的值。 □

Lemma 3. *若在时刻n，有*$band(t_i,n) = band(t_j,n)$*，则在时刻*$n'>n$*，必有*$band(t_i,n') = band(t_j,n')$

Proof. □

5 XBoost的分裂算法

如何在计算上高效地分裂每个节点呢? 定义$D_k^t = \{(x_{ik}, h_i^t)\}_{i=1}^n$，也即是每个样本第$k$个属性上的取值以及第$t$颗树构建完成后损失函数二阶导数在每个样本点处的取值。

$$r_k^t(z) = \frac{1}{\sum_{i=1}^n h_i^t} \sum_{i\in\{i|x_{ik}<z\}}^n h_i^t \tag{18}$$

现在需要求出l个候选分裂点$\{s_{k1}, s_{k2}, \ldots, s_{kl}\}$满足:

$$|s_{k,j} - s_{k,j+1}| < \varepsilon \tag{19}$$

References

[1] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794. ACM, 2016.

[2] Michael Greenwald and Sanjeev Khanna. Space-efficient online computation of quantile summaries. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*, SIGMOD '01, pages 58–66, New York, NY, USA, 2001. ACM.

[3] Edo Liberty Zohar Karnin, Kevin Lang. Optimal quantile approximation in streams. In *Proceedings of 2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 2016.