软件学报 ISSN 1000-9825, CODEN RUXUEW
2024,35(12):5558−5581 [doi: 10.13328/j.cnki.jos.007058] [CSTR: 32375.14.jos.007058]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

基于声明式推理的高效协同查询处理技术*

邱志林[1,2], 寿黎但[1,2], 陈 珂[1,2], 江大伟[1,2], 骆歆远[1,2], 陈 刚[1,2]

[1](浙江大学 计算机科学与技术学院, 浙江 杭州 310027)

[2](区块链与数据安全全国重点实验室 (浙江大学), 浙江 杭州 310027)

通信作者: 陈珂, E-mail: chenk@zju.edu.cn

摘 要: 由于深度学习领域的不断进步, 人们对用协同查询处理 (CQP) 技术扩展关系数据库以处理涉及结构化和非结构化数据的高级分析查询越来越感兴趣. 最先进的 CQP 方法使用用户定义函数 (UDFs) 来实现深度神经网络 (NN) 模型来处理非结构化数据, 并使用关系操作来处理结构化数据. 基于 UDF 的方法简化了查询书写, 允许用户使用单一的 SQL 提交分析查询, 但要求在即席数据分析中能够根据所需性能指标手动选择合适且高效的模型, 这对用户提出了很高的挑战. 为了解决该问题, 提出基于声明式推理函数 (DIF) 的协同查询处理技术, 通过优化模型选择、执行方式、设备绑定等多个查询实现路径构建完整的协同查询处理框架. 基于所提研究设计的成本模型和优化规则, 查询处理器能够计算出不同查询计划的代价, 并自动选择最优的物理查询计划. 在 4 个数据集上的实验结果证实了提出的基于 DIF 的 CQP 方法的有效性和效率.

关键词: 数据库查询优化; 声明式推理函数; 协同查询处理; 模型选择

中图法分类号: TP311

中文引用格式: 邱志林, 寿黎但, 陈珂, 江大伟, 骆歆远, 陈刚. 基于声明式推理的高效协同查询处理技术. 软件学报, 2024, 35(12): 5558–5581. http://www.jos.org.cn/1000-9825/7058.htm

英文引用格式: Qiu ZL, Shou LD, Chen K, Jiang DW, Luo XY, Chen G. Efficient Collaborative Query Processing Technique Based on Declarative Inference. Ruan Jian Xue Bao/Journal of Software, 2024, 35(12): 5558–5581 (in Chinese). http://www.jos.org.cn/1000-9825/7058.htm

Efficient Collaborative Query Processing Technique Based on Declarative Inference

QIU Zhi-Lin[1,2], SHOU Li-Dan[1,2], CHEN Ke[1,2], JIANG Da-Wei[1,2], LUO Xin-Yuan[1,2], CHEN Gang[1,2]

[1](College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China)

[2](State Key Laboratory of Blockchain and Data Security (Zhejiang University), Hangzhou 310027, China)

Abstract: Due to the continuous advancements in the field of deep learning, there is growing interest in extending relational databases with collaborative query processing (CQP) techniques to handle advanced analytical queries involving structured and unstructured data. State-of-the-art CQP methods employ user-defined functions (UDFs) to implement deep neural network (NN) models for processing unstructured data while utilizing relational operations for structured data. UDF-based approaches simplify query composition, allowing users to submit analytical queries with a single SQL statement. However, they require manual selection of appropriate and efficient models based on desired performance metrics during ad-hoc data analysis, posing significant challenges to users. To address this issue, this research proposes a CQP technique based on declarative inference functions (DIF), which constructs a complete CQP framework by optimizing model selection, execution strategies, and device bindings across multiple query execution paths. Leveraging the cost model and optimization rules designed in this study, the query processor is capable of estimating the cost of different query plans and automatically selecting the optimal physical query plan. Experimental results on four datasets validate the effectiveness and efficiency of the proposed DIF-based CQP approach.

Key words: database query optimization; declarative inference function (DIF); collaborative query processing (CQP); model selection

* 基金项目: 国家重点研发计划 (2022YFB3304100); 中央高校基本科研业务费专项资金 (2021FZZX001-24)
收稿时间: 2023-04-12; 修改时间: 2023-06-05; 采用时间: 2023-09-14; jos 在线出版时间: 2024-01-17
CNKI 网络首发时间: 2024-01-18

在数据库中, 可以使用协同查询来同时操作结构化数据和非结构化数据[1]. 协同查询由两部分构成: Q_{db} 和 Q_{learning}. Q_{db} 部分通常用传统数据库处理结构化数据; Q_{learning} 部分通常使用机器学习模型处结构化数据. 一般来说 Q_{learning} 部分中的模型调用会使用 UDF 来实现. 如下查询是基于 UDF 的协同查询:

SELECT agency FROM news WHERE modelA(news.text) IN (‘金融’, ‘财经’);

基于 UDF 的协同查询在 Q_{learning} 部分需要用户指定模型, 如上述查询例子中的 Q_{learning} 部分指定了使用的 UDF 为完成分类任务的具体模型 modelA, 这要求数据分析师等数据库用户对库内的模型知识非常熟悉. 而在这种情况下, 查询的精度和计算成本全由指定的模型来决定, 数据库用户需要根据查询的内容以及性能需求对模型进行人工分析筛选, 若选用模型不当, 将影响协同查询的计算性能. 例如, 一个数据库可能整合了用于分类猫和狗图像的模型 A 和用于分类猫和鸟图像的模型 B. 由于这两个模型提供了不同的分类精度和计算成本, 数据分析师在选择含有猫的数据库记录进行分析时, 需要刻意根据精度指标来选择所需的模型, 并尽量优化计算成本, 这对数据分析师来说是一种挑战. 当库内存在很多模型, 以及这些模型分别包含很多不同的类别时, 要在满足模型精度要求下, 挑选计算成本低的模型或模型组合, 是一个非常有挑战的工作. 具体来说, 在已有的模型库中选择模型组合以满足查询中对任务的类别约束, 同时模型组合的总成本最低, 该问题实际上是一个整数规划问题, 证明如下.

设有 k 个模型为 $\{m_1,\ldots, m_k\}$, 每个模型的决策变量为 x_i, 成本为 c_i. 其中 x_i 如下所示:

$$x_i=\begin{cases}1, & \text{选择模型}m_i,\\ 0, & \text{不选择模型}m_i.\end{cases}$$

任务的总类别数量为 n, 类别约束为 n 维 0-1 向量, 类别对应的变量 p_j 如下所示:

$$p_j=\begin{cases}1, & \text{类别约束包含第}j\text{个类别},\\ 0, & \text{类别约束不包含第}j\text{个类别}.\end{cases}$$

模型 m_i 对应的类别向量 l_i 也是一个 n 维 0-1 向量, 每个类别对应变量 l_{ij} 如下所示:

$$l_{ij}=\begin{cases}1, & \text{模型}m_i\text{覆盖第}j\text{个类别},\\ 0, & \text{模型}m_i\text{不覆盖第}j\text{个类别},\end{cases}$$

则原问题可以表述为如下形式:

$$\begin{cases}\min \sum_{i=1}^{k} c_i x_i\\ \text{s.t.}\begin{cases}\sum_{i=1}^{k} l_{ij} \geqslant p_j \ \forall j=1,\ldots,n\\ x_i\in\{0,1\}\ \forall i=1,\ldots,k\\ p_j\in\{0,1\}\ \forall i=1,\ldots,k\\ l_{ij}\in\{0,1\}\ \forall i=1,\ldots,k;\ \forall j=1,\ldots,n\end{cases}\end{cases}.$$

在基于 UDF 的协同查询中, 除了上述用户选择模型的复杂度之外, 还有模型执行方式和设备选择的复杂度. 基于 UDF 的协同查询在数据库内没有进行相应的优化, 查询优化器会默认使用并行执行的方式运行模型, 并将模型全部部署在某一设备上. 在设备容量受限的情况下, 模型会竞争计算设备的算力, 从而影响查询的执行效率. 对于有多个任务的查询, 查询优化器也不会考虑任务间的相互影响, 从而无法进一步提高查询效率.

因此, 基于 UDF 的协同查询在使用中存在易用性以及性能上的缺陷, 本文考虑基于声明式推理函数 (DIF) 的协同查询以提高协同查询的易用性和计算性能. 基于 DIF 的协同查询由查询谓词和精度约束两部分组成.

(1) 查询谓词包括声明式函数和类别约束, 相比于 UDF, 声明式函数只表明用户需要完成的分类任务, DIF 的具体实现方式则由查询处理器确定, 因此其查询的计算性能由查询处理器来决定.

(2) 精度约束, 在没有指定模型的条件下, 用户需要为查询处理提出精度约束, 以便查询处理器找到同时符合类别和精度约束的模型. 以下是一个基于 DIF 的协同查询中的例子:

SELECT agency FROM news WHERE text_classifier(news.text, 95%) IN (‘金融’, ‘财经’);

查询中的 text_classifier 为 DIF. 这个 DIF 只表明分类任务为新闻文本分类. 在没有指定模型的条件下, 用户需要为查询处理提出类别约束和精度约束. 用户提出的约束可确保查询处理器找到符合约束的模型. 上述查询中 95% 的精度约束表明最终分类结果需要达到 95% 的准确率. 查询中的 ('金融', '财经') 为类别约束, 用于筛选从属于金融和财经的新闻.

在基于 DIF 的协同查询处理中, 最重要的就是如何根据类别约束和精度约束, 确定 DIF 的具体实现方式以尽可能降低查询所需的时间. 本文考虑了 Lu 等人[2]使用的代理模型系列方法, 他们针对需要的类别训练相应的代理模型. 这种代理模型的计算成本远低于原本使用的模型, 故在查询执行时使用代理模型可以大幅度降低查询时间. 但已有的基于代理模型的研究工作存在以下两个缺点. 第一, 在查询处理时需要根据类别约束即时训练相应的代理模型, 增加了训练模型的成本. 第二, 针对类别约束中的每个类别都需要训练一个模型, 在类别数量多时需要的模型数量多, 模型推断时的成本因需要推断的模型数量增多而显著上升. 这两个缺点限制了代理模型方法对于查询性能的优化效果. 同时我们也观察到, 已有工作缺少对模型在不同计算设备上的部署的考量. 而事实上, 计算设备的不同会影响查询时间.

为了改进已有的工作, 本文提出基于声明式推理的高效协同查询处理技术 (DIFCQP). 这一技术会针对分类任务预先训练不同类别集合、精准度和计算时间所对应的模型, 并将模型组成模型库. 查询处理器会在模型库中选取部分合适的模型, 并确定模型部署的位置和执行顺序. 通过预先训练模型, 将模型训练成本从查询成本中进行解耦. 由于训练的模型可以处理多个类别, 从而更少的模型就可以处理同样数量的类别. 模型使用量降低也就减少了推断成本. 通过对模型的部署设备进行考量, 查询处理器可以充分利用多种类的计算设备.

具体来说, 本文的贡献可以分为以下几点.

(1) 本文提出一种基于声明式推理的高效协同查询处理技术 (DIFCQP). 与传统使用 UDF 处理非结构化数据的方法不同, 无需数据分析师指定具体的模型. 只需指定需要满足的性能指标, DIFCQP 即可自动选择合适的机器学习模型及其运行方式, 这一技术大大提高数据分析师的生产力.

(2) 本文提出基于成本估计的优化规则. 首先构建基于声明式推理的协同查询的成本模型. 然后基于模型选择、执行方式和设备绑定等角度分别提出优化规则. 这些规则会选择满足约束条件的、执行成本最低的模型组合并确定模型组合的执行细节. 从而在满足用户要求的前提下最小化目标查询的执行时间.

(3) 本文对提出的查询处理技术进行了大量实验. 结果表明, 在满足用户精度要求的前提下, 可有效降低基于声明式推理的协同查询的反馈时间. 在相同精度要求的条件下, 与 UDF 方式相比, 使用 DIFCQ 的查询时间平均降低 92%; 与 SQL 实现机器学习模型的工作相比, 查询时间平均降低 60%.

1 相关工作

1.1 数据库内机器学习功能

数据库与机器学习这两种技术的结合对双方都有巨大的提升效果. 从人工智能赋能的数据管理[3−7]角度来看, 机器学习模型可以优化数据库的各种部件, 如学习型索引等. 反之, 在引入机器学习功能后, 数据库系统可以在成熟的数据管理之上提供更全面有效的分析能力. 用户在使用该系统时又可以规避单独使用两个系统的交互成本. 李国良等人[8]将以上两种做法定义为 AI4DB 和 DB4AI. 本文主要考虑的是 DB4AI, 即数据库为机器学习模型提供数据, 并更好地支持机器学习功能. 在数据库系统中引入机器学习功能的研究工作可以分为以下 3 类.

第一是将外部实现的机器学习模型包装成 UDF 的形式. 用户可以在 SQL 中调用这些 UDF 来解决问题. 现有的数据库如 Oracle 数据库的 ORE[9]和 Microsoft 的 SQL MLS[10]引入以 R 或 Python 编写的统计学习软件包. MADlib[11]会调用 Python 编写的模型 UDF 来实现简单模型. 复杂的模型会被拆分成多个 SQL 查询. 这些查询需要按照适当的顺序进行执行. 以上基于 UDF 的方案可在各种数据库系统中进行使用, 具备泛用性以及用户查询时的易用性. 本文提到的基于 UDF 的协同查询即使用上述方案进行实现.

第二是部分使用外部机器学习框架, 但对数据库和机器学习框架的交互上进行改进. AIDA[12]构建了 TabularData

对象作为数据库和 Python 机器学习框架的桥梁. TabularData 对象在进行关系型操作时可被视为普通的数据库关系表. 从而可以自然地使用到数据库内的部分优化技术来处理数据. 在涉及机器学习计算时 TabularData 对象又可以被视作 NumPy 的 array. 从而使用 NumPy 的函数来处理数据. MLog[13]是将声明式机器学习 (如 Keras) 与声明式数据管理 (如 SciDB) 结合起来的系统. 它建立在一个类似于 SciDB 的标准数据模型上, 将基础的 n 维张量存成 (n+1) 维的关系表. 然后在数据模型上扩展查询语言为 MLog 语言. 这一类型的方法针对每种特定的机器学习框架, 在数据库中添加数据或查询的转换功能. 这一做法可以降低数据库系统和机器学习框架之间的交互成本. 但这种方案与 UDF 实现方案相比复杂度高且泛用性不足, 需要针对特定的框架进行适应性调整.

第三是直接在数据库内核中实现机器学习模型. Scalable LA[14]引入 VECTOR, MATRIX 等数据类型, 并配套扩展了 VECTOR 和 MATRIX 的乘法等计算. 从而可以用 SQL 去实现机器学习模型. Schüle 等人[15]将模型的训练 (梯度下降) 和推断 (标注) 编写成两个算子. 这两个算子可以像其他传统的算子一样参与查询计划的生成和优化. Lin 等人[1]对数据库中卷积神经网络的推断进行研究. 他们将神经网络的模型参数和输入数据都存储为关系表的形式. 输入数据的每个值依照卷积操作中所在的块位置和块内位置的索引进行存储. 从而可以使用 SQL 来实现卷积和池化的操作, 进而实现卷积网络. 上述方案的优点是充分利用数据库自身的功能. 在消解数据库与机器学习框架间交互成本的同时, 利用查询优化器对模型部分进行优化. 但这一方案存在的问题是, 相比于 UDF 方案中使用成熟的机器学习框架实现的模型, SQL 实现方案要求用户在查询前自己使用 SQL 编写相应的机器学习模型. 这一做法对用户的机器学习能力和数据库能力都有较高的要求, 易用性较差.

1.2 数据库内机器学习的优化

在数据库内引入机器学习功能后就需要考虑降低模型推断的成本. 一方面, 在数据库里模型调用的各个环节都有值得优化的空间. 另一方面就是利用数据库和机器学习任务间的相关关系以缩短使用模型的时间. 这样的工作可分为两类.

第 1 类适用于所有的机器学习模型调用. RETRO[16]针对拥有文本列的数据库, 提前赋予预训练的词向量以避免每次调用模型之前对文本进行转化. Kang 等人[17]的工作则从数据预处理的角度来节省调用模型的成本. 对图像数据进行低分辨率处理可以降低数据预处理的成本. 这一类优化不受查询形式的影响. 但这种优化在泛用性高的同时, 无法利用到查询本身的部分特殊性质.

相较而言, 第 2 类工作则根据模型调用在 SQL 查询中的不同位置和作用, 设计对应的优化方式. 以钮泽平等人的工作[18]为例, 针对选择算子中包含决策树模型的谓词, 提出预筛选+验证的方法, 对输入数据使用初筛规则进行筛选以大幅减少需要决策树模型进行推断的数据量.

与本文关注的查询紧密相关的是利用谓词筛选机制的优化工作. 概率谓词[2]的工作是基于一类特殊谓词展开的. 这类谓词即是二元分类谓词. 谓词中要求分类类别为某一特定类别的数据, 如“$f(T.c)$= ‘Type-a’”. 这一工作对分类任务训练‘Type-a’相应的二分类代理模型. 这种模型只负责判断数据点是否从属于某一类别. 代理模型在体量和推断时间上都要远低于应对所有类别的参考模型, 从而大大降低查询的反馈时间. 这一工作存在的问题是只针对二元分类谓词, 未考虑多类别的情况.

同样是针对二元分类谓词的运行成本降低的问题, 在给定数据、查询的类别以及参考模型之后, NoScope[19]将参考模型应用于数据的一个子集并生成类别标记. 生成的训练数据集可以被用来训练一连串成本更低的模型. 这些模型构成一个级联的模型组合. 在剩余的大多数数据上依照级联模型的置信度逐个运行, 并在置信度达到用户给定的阈值之后停止运行. 从而避免在所有数据上运行参考模型, 大大降低所需的运行成本. 除了二元分类谓词本身的问题之外, 这一方法中级联模型的训练本身即有较大的成本. 级联降低查询时间的效果会受到训练成本引入的影响. 对每个不同的二元分类谓词单独训练出代理模型, 可能获取的查询计划是次优的.

Yang 等人[20]对同一个查询中涉及不同的包含分类任务的谓词时的状况进行利用. 当谓词间存在相关关系, 如“$f(T.c)$= ‘Type-a’ AND $g(T.e)$= ‘Type-b’”中两个谓词的输出结果不独立, 使用两个单独的代理模型分别做推断可能会浪费部分推断的成本. 考虑在线训练代理模型和它们的组合, 分配它们的精度参数, 并将它们放入修改后的查询

计划中. 为了减少训练代理模型的计算开销和延迟, 查询优化器在代理模型构建过程中会重复使用中间结果, 并使用分支和边界搜索来修剪候选查询计划.

上述的方法都是基于代理模型的优化. 这类方法会对查询中给出的类别约束进行利用. 本文将以这一类方法为基础进行改进.

2 基于声明式推理的协同查询

本文考虑的是在关系型数据库中查询的场景. 数据库中关系表 T 同时存在结构化数据和非结构化数据. T 中非结构化数据记为 $T.c$. 用户在挖掘这些数据的信息时, 可以使用基于声明式推理的协同查询. 下文给出这一类查询的形式化定义.

由于数据库中的数据既包含结构化数据, 也具备非结构化数据. 因此相应的数据库查询也需要能够同时处理这两类数据, 这类查询被称为协同查询. 对于非结构化数据的分类, 传统方法会使用特定的机器学习模型 UDF 对数据进行分类. 而声明式推理函数 (DIF) 则有所不同, 它不会指定特定的模型, 而是声明了分类任务并给出精度要求. 其具体定义如下所示.

定义 1. 声明式推理函数 (DIF) 是在数据库中使用的、声明分类任务推理的函数, 其形式为 $y = f(T.c, p)$. 其中 f 是分类任务名称. T 是关系表. $T.c$ 是 T 中的非结构化数据列, 作为 f 的输入. p 是精度阈值, 表明用户需要的最低精度. 返回值 y 是 $T.c$ 对应的分类类别.

以 text_classifier(news.text, 95%) 为例, f = text_classifier 表明分类任务为新闻文本分类. $T.c$ = news.text 表明函数输入为关系表 news 中的 text 列. $p = 95\%$ 表明用户需要分类的准确度至少为 95%.

基于上述协同查询和 DIF 的介绍, 本文提出基于声明式推理的协同查询的定义.

定义 2. 基于声明式推理的协同查询是一种关系查询. 其中选择算子 σ 的谓词包含 DIF. σ 的具体形式如下:

$$\sigma_{f(T.c,p)\ op\ operand\ [\wedge\ldots]}(T),$$

其中, $f(T.c, p)$ 为 DIF, 其计算结果会经过运算符 op 和运算数 $operand$ 的筛选. [∧…] 为可选项, 即允许多个谓词构成合取范式, 其中可能存在部分谓词包含 DIF.

以下面的查询为例:

$$\Pi_{\text{agency}}\left(\sigma_{\text{text\_classifier(news.text, 95\%) IN ('金融', '财经')}}(\text{news})\right),$$

在选择算子中, text_classifier(news.text, 95%) 为 DIF, op = IN, $operand$ = ('金融', '财经'). 表明用户需要输出结果从属于金融和财经两类之一.

当用户发起基于声明式推理的协同查询时, 会将 DIF 的具体实现方式的确定交给查询处理器来解决. 查询处理器在选择实现方式时, 需要考虑的因素为类别约束、精度约束和计算成本.

3 基于声明式推理的协同查询处理框架

当查询编译器接收一个基于声明式推理的协同查询 (约定以 SQL 这样的语言编写查询文本) 后, 需要通过多阶段的处理将查询转换成物理查询计划, 如图 1 所示.

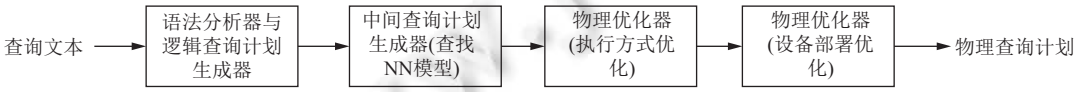


图 1 协同查询处理流程图

下文将对各个阶段的操作进行阐述, 并展示样例查询的处理过程. 样例如下所示:

SELECT agency FROM news WHERE text_classifier(news.text, 95%) IN ('金融', '财经');

第 1 步是将查询转化为初始的逻辑查询计划. 首先对查询进行语法分析, 语法分析器需要将输入的查询文本转换成语法分析树. 然后是将语法分析树转换成初始的逻辑查询计划, 使用关系代数运算符替换语法分析树的节

点和结构, 并产生关系代数表达式. 这一步的两个阶段按照数据库的标准处理流程执行. 上述样例会被转换为表达式:

$$\Pi_{\text{agency}}\left(\sigma_{\text{text\_classifier(news.text, 95\%) IN ('金融','财经')}}(\text{news})\right).$$

第 2 步是将逻辑查询计划转换成中间查询计划. 在初始的逻辑查询计划基础上, 利用代数定律进行等价转化, 从而获得改进后的逻辑查询计划, 称为中间查询计划. 对于选择算子的条件中存在多个 DIF 的情形, 可以根据选择算子相关的分解定律将条件进行拆分, 并考虑使用下推选择等方法优化查询计划. 同时需要为逻辑查询计划中的 DIF 查找合适的模型实现. 选择算子的条件中对 DIF 给出了约束, 即保留 DIF 输出结果在给定范围的部分数据. 依照约束, 查询编译器可以在 DIF 对应任务的模型库中挑选合适的模型集合. 选择的模型集合需要覆盖条件中的类别约束, 满足 DIF 中的精度约束, 并拥有尽可能低的计算成本. 上述样例中, 对于任务 text_classifier, 在满足类别约束 ('金融', '财经') 和精度约束的前提下, 从模型库中挑选低成本的模型来实现 text_classifier. 记选出的模型为 m_1, m_2.

最后使用物理优化器将中间查询计划转换成物理查询计划. 由于中间查询计划中不同算子的物理实现的代价不同, 查询处理器需要执行基于代价的计划选择, 为中间查询计划中的每个算子选择合适的实现方式. 本文考虑的查询类型中比较特殊的部分是确定实现 DIF 的模型的运行方式, 包括执行方式优化和设备部署优化. 在确定具体的运行方式之前, 需要构建与机器学习模型执行相关的代价模型. 而后便可以针对选用的模型计算使用不同设备、不同执行顺序以及不同执行方式 (串、并行) 的代价, 以获取代价最低的运行方式. 模型按照计算出的执行顺序和执行方式依次部署在选中的设备上. 在接收来自其他算子计算后的数据后, 模型就可以完成推理并返回分类类别给需要的算子. 当模型完成计算, 且同一设备有后续模型需要部署时, 使用新模型覆盖旧模型以充分利用该设备的计算能力. 上述样例中, 对于模型 m_1, m_2, 需要计算并行执行和串行执行的成本, 考虑选用哪种执行方式, 并为每个模型选择计算设备, 以达到整体执行成本的最小值.

4 优化准备

在确定了基于声明式推理的协同查询处理框架之后, 针对其中优化查询计划的部分, 提出基于成本估计的优化规则. 本节以及后续章节中将使用一个简单的新闻数据库作为例子进行阐述. 如图 2 所示的新闻数据库中, 包含新闻表、记者表、新闻机构表等关系表格. 新闻表包含发布新闻的日期、新闻的作者和发布机构以及新闻的标题和正文. 记者表和新闻机构表中包含记者和新闻机构的详细信息. 本节内容有两部分: 一是阐述成本模型以供成本估计; 二是对不同模型组合后的整体精度进行估计, 为模型选择提供依据.

新闻ID	发布日期	记者	发布机构	标题	正文
0001	2022/01/01	张三	科技日报社	探秘77 GHz毫米波芯片	中国电科38所发布了一款高性能77 GHz毫米波芯片及模组
0002	2022/01/02	李四	经济日报社	“房住不炒”不应也不会突破	刚刚结束的中央经济工作会议指出, 要确保房地产市场平稳发展
0003	2022/01/03	王五	中国新闻社	旧厂房化身“数字游民”筑梦空间	旧厂房改造的屋舍里, 热腾腾的咖啡和一台台电脑, 代替了轰鸣的机器
…	…	…	…	…	…

记者ID	姓名	所属新闻机构	负责板块
001	张三	科技日报社	教育、科技
002	李四	经济日报社	财经、金融
…	…	…	…

新闻机构ID	机构名称	主要刊物
01	科技日报社	科技日报
02	中国新闻社	中国新闻周刊
…	…	…

图 2 新闻数据库示意图

4.1 成本模型

查询中涉及机器学习模型的部分引入额外的成本. 在正式分析成本的组成之前, 以新闻数据库为例直观表现

引入成本的量级. 考虑以下两个查询, 其一是基于声明式推理的协同查询:

SELECT agency FROM news WHERE text_classifier(news.text, 90%) IN (‘金融’, ‘财经’, ‘科学’, ‘教育’);

为了保证满足上述查询的精度要求, 查询处理器会默认选择复杂、精度高的机器学习模型来完成分类任务. 作为对比, 另一个查询调用了一个计算简单的 UDF. 这个 UDF 不涉及机器学习模型. 查询如下所示:

SELECT agency FROM news WHERE length(news.text) BETWEEN (100, 200);

调节 BETWEEN 的范围, 使上述两条查询具有相近的输出大小. 此时两条查询的查询时间分别为 62 s 和 170 ms. 这个时间对比体现是否使用机器学习模型下查询时间的巨大差别. 从理论的角度来看, 数据库查询由多个算子嵌套执行. 普通的算子计算通常使用“=, +, −, *, /”等简单的方式来处理一条数据. 而带有机器学习模型的算子, 在处理一条数据时需要几百、几千次的乘法、加法计算. 这种计算量上的差别最终导致基于声明式推理的协同查询的响应时间远超过不涉及机器学习模型的查询. 因此查询处理器在估算成本时, 需要考虑机器学习模型推断相关的成本, 并权衡机器学习模型成本与传统算子成本之间的关系. 从而优化器在比较不同查询计划的成本后可以选出实际成本更低的计划.

加入机器学习能力后新引入的成本穿插在数据库调用模型整个过程的各个环节. 模型的成本分析分为 3 部分: 模型部署、数据传输及模型推断. 在分析具体成本之前, 考虑前文提到的新闻文本分类任务. 假定确定了提前训练好的 RNN 分类模型, 可以部署模型的 GPU 设备, 以及单条新闻文本的最大存储占用. 最大文本长度设为 600 个字, 而每个字按照 Unicode 进行存储, 则会占用 600×3=1800 B 的存储空间.

概念 1. 模型部署成本是模型从存储设备读取并转移到设备上的成本. 给定模型 m 以及对应的设备 d, 模型部署成本为 $t^{\text{dep}}_{m,d}$.

模型部署方面, 需要考虑设备的差别. 为了充分利用设备, 需要考虑将模型部署到不同设备上. 由于设备本身的差异以及设备与数据库所在设备的交流方式不同, 所以不同设备上部署的成本也会不同. 模型在不同设备的部署成本会存储在配置文件. 优化器通过读取配置文件, 可以参考里面的估算时间进行查询计划的制定. 以新闻数据库为例, m 是 RNN 分类模型, d 是 GPU 设备. 在查询时需要将模型从文件加载到内存中, 再利用 CUDA 的 deploy 指令将模型部署到 GPU 上. 从而模型可以在 GPU 上进行计算. 部署所花费的时间便是 $t^{\text{dep}}_{\text{RNN,GPU}}$.

概念 2. 数据传输成本是模型所需数据从数据库中读取并转移到设备上的成本. 给定设备 d 的传输速率 v^{trs}_{d} 和模型 m 需要的数据量 b, 模型的数据传输成本为 $t^{\text{trs}}_{d}=\dfrac{b}{v^{\text{trs}}_{d}}$.

数据传输方面, 需要考虑设备的差别以及数据本身的影响. 由于非结构化数据的种类繁多, 不同种类的数据占用空间不同. 同样数据量的视频就比图像需要传输的数据包大小更大, 从而也会影响调用模型时需要花费的时间. 同样以新闻数据库为例, 在 GPU 上完成新闻文本分类, 需要把数据传输到 GPU 上. 机器学习模型通常以 batch 的形式进行计算的. 假设有 64 000 条新闻文本需要分类, 那么按照每个 batch 有 64 条新闻文本的大小将文本切分为 1 000 个 batch. 这其中每个 batch 的数据块大小 block_size = 64×1800 B = 112.5 KB. 则 b = 112.5 MB. 假定 $v^{\text{trs}}_{\text{GPU}}$ 为 100 MB/s, 则数据传输成本为 $t^{\text{trs}}_{\text{GPU}}=\dfrac{112.5\ \text{MB}}{100\ \text{MB/s}}=1.125\ \text{s}$.

概念 3. 模型推断成本是模型在设备上对数据进行推断的成本. 给定模型 m、设备 d 和数据量 b, 有推断速率 $v^{\text{inf}}_{m,d}$, 则模型推断成本为 $t^{\text{inf}}_{m,d}=\dfrac{b}{v^{\text{inf}}_{m,d}}$.

相比于前两者来说, 模型推断成本通常在总时间成本中占据更大的份额, 是性能问题的主要瓶颈.

在新闻数据库查询中, RNN 分类模型和新闻文本均在 GPU 上时, 便可以用模型进行分类得到类别结果, 其推断成本为 $t^{\text{inf}}_{\text{RNN,GPU}}$.

考虑某一个模型集合为 $M=m_i \mid i=1,\ldots,n$. 集合中模型部署的设备位置为 $D=d_i \mid i=1,\ldots,n$. 模型对应需要处理的输入数据的 batch 数量为 $B=b_i \mid i=1,\ldots,n$. 在新闻数据库查询中, 假设最后选取了两个分类模型 m_1、m_2 分别用来分类财经、金融类新闻和科技、教育类新闻 (将这些模型组合以实现这一任务的可行性将在后续解释), 并且将 m_1 部署到 d_1 = GPU 上, m_2 部署到 d_2 = CPU 上. 假定两个模型需要的输入 batch 数量相同, $b_1=b_2=1000$.

概念 4. 任务部署成本是任务对应的模型集合的部署成本总和 C_{dep}, 其计算如下所示:

$$C_{\text{dep}} = \sum_{i=1,\ldots,n} t^{\text{dep}}_{mi,di} \tag{1}$$

概念 5. 任务传输成本是任务对应的模型集合所需数据传输成本总和 C_{trs}, 其计算如下所示:

$$C_{\text{trs}} = \sum_{i=1,\ldots,n} t^{\text{trs}}_{di} \tag{2}$$

概念 6. 任务推断成本是任务对应的模型集合完成推断的成本总和 C_{inf}. 在模型分段为 $M = m_i \mid i = 1,\ldots, k$, 其中模型段 $M_i = m_{ij} \mid j = 1,\ldots, n_i$ 时, 成本计算如下所示:

$$c_i = \max_{j=1,\ldots,n_i} \left(t^{\text{inf}}_{m_{ij},d_{ij}} b_{ij} \right) \tag{3}$$

$$C_{\text{inf}} = \sum_{i=1,\ldots,n} c_i \tag{4}$$

对于模型推断成本 C_{inf} 中的模型分段, 在不同设备上部署模型的优势在于能够同时运行多个模型, 从而提高执行效率. 此外, 模型的成本计算方式随着它们的执行方式和顺序变化而变化. 将查询计划考虑的模型集合划分为模型段. 每一个模型段中的模型并行执行. 段与段之间的则是串行执行. 为了解释这种划分的产生, 将上文的新闻数据库查询的感兴趣类别的数量进行扩大, 变成如下的形式:

SELECT agency FROM news WHERE text_classifier(news.text, 90%) IN (‘财经’, ‘金融’, ‘科技’, ‘教育’, ‘时尚’, ‘家居’);

假设最后选择了 3 个模型 m_1, m_2, m_3 分别对应于其中两个类别, 并且假设新闻表中财经类和金融类的新闻的占比偏多, 如 40%, 则可以把 3 个模型分成两个模型段: 第 1 段只有 m_1, 它会完成财经、金融类的分类, 并为后续的模型留下 60% 的数据; 第 2 段的 m_2 和 m_3 可以在不同设备上同时分类剩下的 600 个 batch 的数据, 而不是完整的 1 000 个 batch. 而这种划分只是其中的一种方式, 具体的划分由后续优化规则确定.

将 3 部分的时间成本汇总得到机器学习模型的总成本. 如下所示:

$$C = C_{\text{dep}} + C_{\text{trs}} + C_{\text{inf}} \tag{5}$$

在本文后续的章节中, 将成本的计算简写为关于模型 m, 设备 d 及 batch 数量 b 的函数, 即 $Cost(m, d, b)$.

4.2 精度估计的理论依据

人们在实际使用机器学习模型时, 不会追求每个数据点的准确推断, 而是对数据整体准确性进行要求. 倾向于性能的用户会对性能和精度分别提出要求. 对于性能, 用户需要查询性能尽可能高. 对于精度, 用户会提出对准确度的接受阈值 t. 模型精度需要达到用户提出的阈值才会被接受.

模型在应用于实际数据进行推断之前, 它的实际精度是无法获知的. 因此只能利用模型在历史数据上推断的准确率来估算模型的精度. 当需要挑选一个模型集合来共同实现分类任务时, 就需要利用集合中各模型的精度来估算模型组合的精度. 详细来说, 假定用户针对任务 q 提出的类别约束为 L_q, 选出的模型组合为 $M = \{m_1,\ldots, m_k\}$. 模型组合对应的类别集合 $L = \{L_1,\ldots, L_k\}$ 足以覆盖 L_q. 从而模型组合满足类别约束:

$$L_q = \cup_{i=1,\ldots,k} l_i \tag{6}$$

以下面的查询为例. 对于文本分类任务 q, 类别约束 L_q 就是指 (‘财经’, ‘金融’, ‘科技’, ‘教育’) 这个所需的新闻类别的范围:

SELECT agency FROM news WHERE text_classifier(news.text, 90%) IN (‘财经’, ‘金融’, ‘科技’, ‘教育’);

假定最后选出来的两个模型为 m_1、m_2. 这两个模型分别对应类别约束 l_1 = (‘财经’, ‘金融’, ‘科技’, ‘其他类’) 和 l_2 = (‘科技’, ‘教育’, ‘娱乐’, ‘其他类’). 那么 $L_q \subset l_1 \cup l_2$, 这表明两个模型足以满足查询者的类别要求. L_1 和 L_2 中都包含的其他类, 保证模型找不到新闻的具体类别时也会有输出结果. 至于精度, 以 L_1 为例, 假定在历史数据上收集到财经、金融、科技和其他类的准确率分别为 90%、90%、95%、85%; 召回率分别为 95%、95%、85%、90%. 那么 m_1 每分类 100 条财经新闻, 就有 90 条是判对的. 而新闻表里每 100 条科技新闻, 就有 85 条被成功选出.

模型 m_i 的类别约束 l_i 会包含 $|l_i| - 1$ 个分类任务 q 中的实际类别及一个“其他”类别. 这里的“其他”类别确保模型不会把其余类别的数据错误分类. m_i 在每个类别 l_i^j 上的准确率和召回率分别为 $P_i = \{p_i^1,\ldots, p_i^k\}$ 和 R_i =

$\{r_i^1,\ldots,r_i^k\}$. 基于这些数据, 可以考虑一个大小为 N 的数据集. 其中某一个类别 $c\in L_q$ 所对应的数据量为 N_c. 对实际从属于类别 c 中的一个数据, n 个模型中的 k 个返回非“其他”的结果可能性为公式 (7) 所示:

$$r(n,k)=\sum_{T\subset M,|T|=k}\left(\prod_{i\in T}(1-r_i^o)\prod_{j\notin T}r_j^o\right) \tag{7}$$

从而模型集合 M 整体对类别 c 的召回率为:

$$R_c=\sum_{k=1}^{n}\frac{1}{k}r_i^c r(n-1,k-1) \tag{8}$$

如果按照召回率进行 M 的准确率计算的话, 模型集合的准确率为:

$$R=\sum_{c\in L}\frac{N_c R_c}{N} \tag{9}$$

如果 M 中所有的模型在 L_q 内的所有类别上的召回率均满足用户提出的精度约束 r, 即 $\forall 1\leqslant i\leqslant n, c\in L, r_i^c\geqslant r$, 且 r 满足 $r\geqslant\frac{n-k}{n}$. 那么就可以保证 $r(n, k)$ 的取值范围, 即:

$$r(n,k)\geqslant r(n,k,r)=\binom{n}{k}r^{n-k}(1-r)^k \tag{10}$$

从而 R_c 的取值范围也可以被固定在 r 的范围内, 如下所示:

$$R_c\geqslant\sum_{k=1}^{n}\frac{r}{k}r(n-1,k-1,r)=\sum_{k=1}^{n}\frac{r}{k}\binom{n-1}{k-1}r^{n-k}(1-r)^{k-1}=\frac{r}{1-r}(1-r^n)\approx\frac{r}{1-r} \tag{11}$$

从而 $R\geqslant r\sum_{c\in L}\frac{N_c}{N}=r$, 也就是说模型集合 M 的整体召回率满足精度阈值 r. 总之, 当模型集合中模型的召回率满足要求时, 模型集合整体的召回率便可以满足用户的需求.

对于准确率的估算, 假定用户给定的准确率阈值为 p. 准确预测类别 c 的模型 m_i 在类别 c 上的准确率为 p_i^c. 那么模型组合 M 对于类别 c 的预测准确率为 $P_c=p_i^c$. 记 $Pred_c$ 为模型组合判定为类别 c 的数据量. 则模型组合的准确率可以由公式 (12) 进行计算:

$$P=\sum_{c\in L}Pred_c\frac{P_c}{N}\geqslant p\sum_{c\in L}\frac{Pred_c}{N}=p \tag{12}$$

综上所述, 当模型在目标类别的准确率都能满足阈值 p 时, 模型集合的准确率便可以满足用户的精度需求.

5 单分类任务的优化规则

本节考虑查询只包含一个分类任务时的优化, 并将基于模型选择、执行方式和设备绑定这 3 个方面提出优化规则.

5.1 基于模型选择的优化

基于模型选择的优化规则: 给定分类任务 q、类别约束 L_q、q 的备选模型库 M、精度要求 t. 在 M 中选出子集 M'以满足 L_q 和 t, 同时 M' 的计算成本最低.

当查询处理器收到查询时, 它会确定具体任务 q 和相应的类别集 L_q. 查询前会预先针对任务 q 训练一些模型并组成集合 $M_q=\{M_1,\ldots,M_n\}$. M_q 中的模型对应的标签集为 $L_q=\{l_1,\ldots,l_n\}$. 如图 3 所示的样例中分别给出了目标类别以及一些模型所对应的标签集. 如模型 m_1 可以分类集合 $l_1=$ (‘财经’, ‘金融’, ‘其他’); m_2 可以分类集合 $l_2=$ (‘教育’, ‘家居’, ‘其他’) 等. 图 3 中模型的目标类别也会有不同程度的重合. 查询处理器可以从这些模型中挑选不同的模型子集 $M'\subset M$, 使得他们对应的类别 L'可以覆盖 L_q, 即对于 $L'=\cup_{m\in M'}L_m$, 有 $L_q\subset L'$.

以新闻数据库的查询来进行阐述. 给定以下的查询, $L_q=$ (‘财经’, ‘金融’, ‘科技’, ‘教育’, ‘娱乐’, ‘家居’):

SELECT agency FROM news WHERE text_classifier(news.text, 90%) IN (‘财经’, ‘金融’, ‘科技’, ‘教育’, ‘娱乐’, ‘家居’);

如图 3 所示的目标类别就是 L_q. 而备选模型则选取了模型库中与 L_q 有交集的模型. 每个模型都列举了它的类别集合. 在不考虑成本和精度时, 为了覆盖目标的类别集合 L_q, 可以使用 m_1, m_3, m_4 进行组合, 也可以选择 m_1, m_2, m_5. 而在这样的两套组合中进行选取, 就依赖于对二者的精度和成本的权衡.

目标类别: 财经、金融、科技、教育、娱乐、家居

模型	类别	精度
m_1	财经、金融、其他	(0.9,0.8)、(0.9,0.8)、(0.95,0.9)
m_2	教育、家居、其他	(0.85,0.9)、(0.9,0.8)、(0.9,0.8)
m_3	金融、科技、教育、其他	...
m_4	娱乐、家居、其他	...
m_5	科技、娱乐、其他	...

图 3 备选模型的示意图

除分类任务 q 和类别约束 L_q 之外, 查询中会指定需要达到的精度阈值 t. 上述查询中的“text_classifier (news.text, 90%)”就表明查询者希望分类的精度达到 90% 以上. 常见的精确度指标有准确度 (Precision, P)、召回率 (Recall, R)、$F1$ 分数等. 在模型训练的同时会采集模型的准确率 $P=\{p_1,\ldots,p_n\}$和召回率 $R=\{r_1,\ldots,r_n\}$. 仍以图 3 为例, 图中只显示 m_1、m_2 的数据, 其他模型以此类推. 在历史数据上采集的 $p_1=\{0.9, 0.9, 0.95\}$, 而 $r_1=\{0.8, 0.8, 0.9\}$.

在统计意义下, 只要模型集合中每个模型在选定的指标下均能满足阈值条件, 则该集合可以满足精确度的要求. 不失一般性的, 假定默认指标为准确率 p. 设定精度阈值 t 之后, 可以先对任务 q 对应的模型集合 M_q 进行筛选, 并将精度指标低于阈值的先过滤掉. 这样做可以减少后续模型选择时需要考虑的模型范围, 降低优化成本. 如模型 m_1, 查看 p_1 即可发现这个模型是满足约束的. 而 m_2 模型的“教育”类别统计的准确率只有 0.85. 模型在精度不满足要求时不会被选中.

在本节中假定模型均部署在同一设备 d 上. 从而在给定模型 m 和对应输入数据的 batch 数量 b 之后, 成本函数为 $Cost(m, b)=Cost(m, d, b)$. 在这一假设下, 设备可以同时容纳挑选的模型子集 M'中的所有模型. 并且模型之间不会互相影响. 从而可以约定这一规则的优化目标: 首先确保对于每个标签 $l\in L'$来说, M'的准确性能够满足用户的需要; 其次, 尽量降低 M'的成本.

在不考虑模型顺序的前提下, 寻找模型集合 M'的问题与集合覆盖问题比较类似. 对这个问题, 考虑所有的模型组合并挑选出最优的集合的时间复杂度是指数级的, 即 $O(2^n)$. 在模型数量相对较少时, 可以考虑将所有涉及 L 内类别的模型预先选出, 并比较它们的所有组合以获得最优的模型集合. 但模型数量的增长会使得优化过程变得极其漫长. 此时就需要近似算法加以解决.

参考集合覆盖问题的贪婪搜索算法, 把贪婪搜索算法加以修改并用在本文的问题场景中. 具体来说, 按如下公式计算模型成本, 其中 C 为已考虑的类别集合:

$$cost[i]=\frac{Cost_d(m_i,b)}{\left|count(l_i^q)\setminus C\right|} \tag{13}$$

其中, 模型成本 $cost[i]$ 的选取基于这样的考量: 模型本身的计算成本 (公式 (13) 中的分子) 越低, 它覆盖的未考虑类别数量 (公式 (13) 中的分母) 越多, 那么这个模型的成本越低, 应该越早使用.

算法的流程如下: 每次从备选集之外的模型中选择成本最低的模型加入备选集, 直至覆盖类别约束. 这种方式的计算复杂度为 $O(n)$. 这一方法的优化成本非常低, 但它探索的模型集合范围非常小. 最终导致获取的模型组合可能与最优解有较大的差距. 以新闻数据库查询为例, 前文提到可以选出 m_1, m_3, m_4 和 m_1, m_2, m_5 两组模型组合来满足“text_classifier(news.text, 90%) IN (‘财经’, ‘金融’, ‘科技’, ‘教育’, ‘娱乐’, ‘家居’)”的要求. 假设 5 个模型的成本分别为 10, 20, 25, 25, 40. 贪婪搜索的算法在选完最低成本的 m_1 后接着会选择 m_2 和 m_5, 获得总成本 10+20+40=70 的模型组合. 而另一个没有被选中的组合 10+25+25=60 的总成本反而更低. 上述例子体现了贪婪搜索算法的不足.

为了解决贪婪搜索算法搜索空间过小的问题, 考虑使用具有贪婪特性并有更大搜索空间的集束算法来进行改良. 给定集束的大小 bs. 先将集束拥有的模型、类别集合与成本进行初始化. 在每一个搜索阶段, 基于集束中的每一条搜索路径, 计算引入未考虑模型的成本. 按成本从低到高排序, 便于选出前 bs 个模型加入对应路径的备选模型, 从而构建 bs 条新的备选路径. 由于集束中有 bs 条搜索路径, 每条路径有 bs 条对应的备选路径, 因此会获得共计 $bs\times bs$ 条备选路径. 将这些备选路径按成本从低到高排序, 并选出前 bs 条作为新的集束, 周而复始. 直到集束中有搜

索路径 (模型集合) 可以满足用户的类别约束. 此时便可以把不同搜索路径中运行成本最低的作为搜索结果进行返回. 仍以新闻数据库为例, 在贪婪搜索算法的使用下选择了 m_1, m_3, m_4 这一组模型. 改成大小为 2 的集束进行搜索. 在选择了 m_1 之后, 算法会保留 m_2 和 m_3 两条路径. 沿着这两条路径分别加入 m_5 和 m_4. 在这两条路径的对比中, 发现路径 $m_1 \rightarrow m_3 \rightarrow m_4$ 的成本更低, 从而最后保留该路径对应的模型组合. 这体现了集束搜索扩大搜索空间的优势.

集束搜索扩大搜索空间的程度取决于集束的大小. 集束太小则一开始就会忽略掉可能加入最优模型集合的模型, 但集束太大会带来很大的优化成本. 所以在实际使用过程中, 会动态调节集束的大小. 即起初使用大集束以扩大搜索范围. 在不断加入模型之后, 慢慢地将集束进行缩小, 以降低整个优化过程中的成本.

在基于模型选择的优化规则中, 优化收益来源于选择的模型集合总体的计算成本相比于全类别分类的复杂模型计算成本的下降. 记查询中对分类任务 q 的类别约束为 L, 其大小为$|L|$, 对应的全分类模型的计算成本为 C, 对应大小为 N^q 模型库中的模型平均计算成本为 $\bar{c}$, 平均覆盖类别数量为 l. 则优化收益为 $profit_1 = C - \frac{|L|}{l}\bar{c}$. 而优化成本则是搜索过程所需花费的成本, 对集束搜索来说, 通过历史记录可知每次计算模型成本的代价为 c, 则优化成本为 $cost_1 = N^q \log N^q \frac{|L|}{l} c$. 比较收益 $profit_1$ 和成本 $cost_1$ 即可判断是否使用该规则.

5.2 基于执行方式的优化

单分类任务下基于模型执行方式的优化规则: 对实现分类任务 q 的模型组合 M, 选取并行/串行执行方式中成本较低的方式来执行.

在第 5.1 节中并未刻意考虑模型的并行/串行运行的问题. 不管是贪婪搜索还是集束搜索算法, 在计算引入的模型成本时, 都需要考虑到已处理模型的覆盖类别, 并尽量找到覆盖更多未考虑类别的、且成本较低的模型. 这种做法更偏向于找到顺序执行的模型序列. 并行执行需要考虑的方向有所不同. 本节将对模型执行的不同方式进行分析和比较.

模型的执行可以分为并行和串行两种方式. 对于并行执行的方式, 不同模型需要对同样的全量数据进行推断, 但这些模型可以在同一设备或不同设备上同时运行. 最后将不同模型的结果进行合并得到目标结果. 一方面这种运行方式不需要确定模型的执行顺序, 只需要把应该参与运行的模型挑选出来即可; 另一方面, 这种运行方式的查询反馈速度取决于运行时间最长的模型. 假定接受同样 batch 数量 B, 则查找目标如下所示:

$$M' = \arg\min_{M' \subset M} \max_{m \in M'} Cost_d(m, B) \tag{14}$$

而对于串行执行的方式, 不仅要选择模型集合 M', 还需要计算 M'中模型的执行顺序. 从而模型可以按照计算出来的顺序依次执行. 在执行细节上, 前一个模型在它的输入数据之上完成推断工作, 获取输出类别. 将输出类别与约束类别集 L_q 作比较, 从而可以把符合要求的数据移除以减少下一个模型需要处理的数据的量. 这种运行方式的查询时间即为顺序执行模型推断的时间之和. 在查找串行执行的模型集合时, 体现串行性质的变量主要是模型的输入数据量, 同一模型集合不同顺序下模型需要处理的数据量不同. 假定每个模型 m 接受的数据 batch 量为 B_m, 则查找目标如下所示:

$$M' = \arg\min_{M' \subset M} \sum_{m \in M'} Cost_d(m, B_m) \tag{15}$$

贪婪搜索对成本的计算和对模型的选取实际上遵从两个原则: 计算成本尽可能低; 覆盖尽可能多的未考虑的类别. 这样的做法更倾向于顺序执行. 而对于并行执行来讲, 模型本身的计算成本相比于覆盖的类别数量来讲更加重要. 对集束搜索来说, 每个阶段进行成本计算时与贪婪搜索一样会考虑前面模型覆盖的类别. 同时, 在计算集束中每条搜索路径的累计成本时, 也会考虑模型筛选之后的数据量. 从而计算的模型组合成本更接近顺序执行时的成本.

下文的新闻数据库查询的例子可以解释集束搜索在串行执行上的偏向性:

SELECT agency FROM news WHERE text_classifier(news.text, 90%) IN (‘财经’, ‘金融’, ‘科技’, ‘教育’, ‘娱乐’, ‘家居’);

假定有另一组已训练好的模型{m_1,…, m_5}. 这组模型中 m_1, m_2, m_3 分别对应于类别集: (‘财经’, ‘金融’), (‘科技’, ‘教育’) 和 (‘娱乐’, ‘家居’). 而 m_4, m_5 则分别对应类别集: (‘财经’, ‘金融’, ‘科技’) 和 (‘教育’, ‘娱乐’, ‘家居’). 假

定模型对应的成本为 10, 10, 10, 14, 14. 就集束搜索来说, 选择$\{m_4, m_5\}$作为目标模型集在串行计算上的成本为 14+14=28<30, 成本更低. 但就并行执行来说, 模型集$\{m_1, m_2, m_3\}$的成本为 10, 比成本为 14 的模型集$\{m_4, m_5\}$的计算时间更少.

综上所述, 在第 5.1 节的模型搜索方法之外, 还需要找到更好的搜索并行执行模型的算法. 在同样设备 d 和同样的输入数据 batch 数量 b 的情况, 计算出各模型的运行成本. 按成本从低到高排列模型并逐一考察模型. 当模型的类别集包含未考虑类别时, 将该模型添加进模型集中. 满足类别约束 L 时的模型集就是并行执行最优的模型集. 在对比串行和并行两种方式各自的最优模型集之后, 便可以确定选用其中哪种方案.

基于执行方式的优化实际是对两种执行方式进行比较. 在不采用该规则的情况下默认使用其中一种执行方式, 则优化收益应为两种方式的预估成本差值. 在第 5.1 节中基于模型选择的优化过程可以获得串行执行的预估计算成本 C_1. 而对于并行执行, 在未执行优化之前无法获知其成本, 只能用模型库中模型平均成本来作为估计值, 即 $\bar{c}$, 则优化收益为$profit_2 = |C_1 - \bar{c}|$. 优化成本是查找并行执行模型集合的预估成本, 即$cost_2 = N^q c$. 比较收益 $profit_2$ 和成本 $cost_2$ 即可判断是否使用该规则.

5.3 基于设备绑定的优化

单分类任务下基于设备绑定的优化规则: 对模型集合 M 中任一模型 m, 找到合适的设备 d 进行绑定.

不同的设备会影响在设备上部署模型、将输入数据传输到设备以及模型推断的时间. 对于单个模型 m, 使用成本模型可以计算将其绑定到所有设备上的成本. 从而很容易选择可容纳 m 并且成本最小的设备来让 m 执行推断任务.

当涉及多个模型时, 情况则有所不同. 第一, 不同模型在同一设备上会有不同的计算成本. 如果模型和计算设备都不同, 他们形成的多种组合又会有不同的成本. 第二, 设备有不同的存储容量, 它能存储的模型数量和种类也不同. 第三, 在同一设备上部署的模型过多时会出现竞争算力的情况. 第四, 由于模型在执行方式上有串行和并行的区分, 在不同时刻模型可能需要分配到不同的设备上. 基于以上考量, 模型的绑定位置需要被仔细研究.

考察某个在新闻数据库中进行查询的分析员. 他拥有的计算设备有算力低、内存大的 CPU 设备, 也有算力较高, 但内存容量较少的 GPU. 另外可能还有算力和内存都在 CPU 和 GPU 之间的嵌入式设备. GPU 的优势是算力最高. 它的劣势是内存不足, 并且它的算力也会被多个模型竞争使用. 因此所有模型在 GPU 运行是不合适的. 同时其他设备空置也造成计算资源的浪费. 所以在面对复杂的模型组合时, 还是会考虑将不同模型部署在不同的设备上.

系统拥有的计算设备记为 $D = \{d_j \mid j = 1,\ldots, m\}$. 对于 D 中每个设备 d 来说, 它有对应的剩余容量计算方式 $Remain_d(m, M_{\text{in}})$. 即在设备 d 上已有 M_{in} 这些模型部署时, 引入 m 之后的剩余容量. 从存储的容量角度上看, 模型 m 本身的体量 $size_m$ 以及它处理的数据占用 $data_m$ 之和应该小于设备的剩余存储空间, 这样才能保证模型的成功部署. 其中数据的占用由 batch size 和数据的类型决定. 而从算力的容量角度上看, 在新模型加入之后, 首先分配给已有模型 M_{in} 的算力变动不应过大, 其次分配给新模型 m 的算力应接近单独部署的算力. 这样可以尽可能降低该设备上的模型推断时的互相影响. 综合这两个角度, 只有在 $Remain_d(m, M_{\text{in}}) \geqslant 0$ 时可以将新模型 m 部署到设备 d 上. 两种执行方式对应的模型组合分别为: 串行的模型组合 (按顺序排列) $M_s = \{m_1^s, \ldots, m_k^s\}$ 和并行的模型组合 (顺序无关) $M_p = \{m_1^p, \ldots, m_n^p\}$. 下文将分析这两种执行方式的设备绑定.

对于串行组合 M_s 来说, 模型是按顺序逐个执行的. 因此每个模型的设备选择是相对独立的. 在提供了可用设备集合 D 对于每个模型 m_i^s 以及对应的 batch 数量 b_i^s 之后, 利用前文提到的成本模型可以计算这个模型在各个设备 d_j 上的成本 $Cost(m_i^s, d_j, b_i^s)$. 从而比较统一模型在不同设备的成本以决定选择什么设备.

对于并行组合 M_p 来说, 模型可并行执行. 因此单个模型的设备选择会影响剩下的其他模型的设备选择. 算法 1 展示了并行执行的设备选择算法. 首先在第 1, 2 行中计算每个模型在每个设备上的预估计算成本. 然后在第 3 行将每个模型对应的设备按成本从低到高排列. 由于每个模型都有成本最低的设备, 从而可以将模型按最低成本从高到低排列. 在第 4–8 行中, 从最高计算成本的模型开始, 将该模型部署在其对应成本最低的、可容纳模型的设备上. 对后续的模型按顺序处理. 具体来说, 按照模型的设备顺序进行排查. 当该设备足以容纳模型时, 将模型部署到该设备上. 第 9 行中, 将每个设备上的模型按照模型下标为键, 设备下标为值存储为字典输出为结果.

算法 1. 并行执行的设备选择算法.

Input: 模型的集合 $M^q = \{m_1,\ldots, m_n\}$, 设备的集合 $D = \{d_1,\ldots, d_k\}$, 数据量 B 以及剩余容量计算公式 $Remain_d(m, M)$;
Output: 最终每个模型对应选择的设备, 记为 $Map = \{m_1: d_1', \ldots\}$.

1. **for** $i = 1,\ldots, n$ **do**
2. 计算模型 m_i 在各设备 d_j 上的计算成本并将各设备按成本升序排列
3. 将个模型按在不同设备上的最低成本降序排列
4. **for** $i = 1,\ldots, n$ **do**
5. **for** $j = 1,\ldots, k$ **do**
6. **if** 设备 d_j 的剩余容量足以容纳模型 m_i **then**
7. $Map[m_i] = d_j$
8. break
9. **return** *Map*

以新闻数据库查询为例, 假定在模型选择后得到 3 个模型 m_1, m_2, m_3. 他们在同一数据量 B, 在同一 CPU 上的成本分别为 10, 20, 30. 其中模型部署和数据传输的成本分别为 2, 2.5, 3. 假设分析员拥有的 GPU, FPGA 的计算能力分别为 CPU 的 3 倍和 2 倍, 但模型部署和数据传输的成本相比 CPU 也增长了, 分别为 CPU 的 3 倍和 2 倍. 那么把 m_1, m_2, m_3 分别部署在 CPU, FPGA, GPU 上执行的成本分别为 10, 13.75, 18. 此时并行执行的成本为 18, 各种设备都被充分利用起来. 而如果全部部署在 GPU 上, 且 GPU 的容量不足时, 模型只能逐一执行, 此时成本为 8.7+13.3+18=40, 比起分散部署的成本更高.

综合上述方法, 可以使用串行/并行两种运行方式相应的方法来确定最终每个模型绑定的设备. 然后考量两种方式各自的总成本, 以决定最终使用哪种计划进行执行.

基于设备绑定的优化规则中, 在不采用该规则的情况下默认使用同一设备 d. 记设备数量为 N^d, 则优化成本为 $cost_3 = N^d c\frac{|L|}{l}$. 对优化收益的计算需要分两种执行方式. 对串行执行来说, 在优化之前无法得知每个模型对应计算成本最低的设备, 只能用模型在各设备上的计算成本平均值 $\overline{c_m}$ 作为估计值. 优化收益为 $profit_3 = \sum_m \min(c_{md} - \overline{c_m}, 0)$, 即对于每个模型来说, 如果设备 d 上的成本更低, 则没有收益; d 上成本更高, 则估计收益为 $c_{md} - \overline{c_m}$. 对并行执行来说, 记模型 m 的体量为 $size_m$, 设备 d 上的容量为 R_d, 各模型在设备 d 上的平均计算成本为 $\overline{c_d}$, 各模型在各设备上的平均成本为 $\hat{c}$, 则优化收益为. $profit_3 = \frac{\sum_m size_m}{R_d}\overline{c_d} - \hat{c}$. 对不同执行方式比较收益 $profit_3$ 和成本 $cost_3$, 并判断是否使用规则.

6 多分类任务的优化规则

将前文使用到的新闻数据库查询进行扩展, 得到以下的查询:

SELECT agency FROM news WHERE text_classifier(news.text, 90%) IN (‘财经’, ‘金融’, ‘科技’, ‘教育’, ‘娱乐’, ‘家居’) AND entity_classifier(news.title, 80%) IN (‘组织’, ‘地址’, ‘姓名’, ‘书名’);

这个查询中添加了一个分类任务. 查询中的任务之间以 AND 进行连接. 在这个查询里, 两个分类任务的输出结果之间存在一定的相关性. 比如正文是家居类别的, 其标题很有可能带有地址类别. 正文是教育类的, 标题可能容易判别出组织类别, 等.

本节考虑多任务查询. 查询中每一组任务及约束记为 (q_i, L_i). 任务之间的关系为 $(q_1, L_1)\&\ldots\&(q_m, L_m)$, 即合取各任务对应的谓词来获取最终的结果. 在引入了多任务后, 查询优化遇到了新的挑战. 一方面不同任务之间的类别可能存在一定的相关性, 比如说任务 A 的类别 a、b 的数据可能会与任务 B 的 c、d、e 类别存在足够多的覆盖关

系. 另一方面不同任务之间的串行/并行关系以及各自的先后关系也有更多可能的选项.

当用户提出的查询包含多个分类任务及相应的类别约束时, 本文基于模型选择、执行方式、设备绑定这 3 方面进行优化.

6.1 基于模型选择的优化

多分类任务下基于模型选择的优化规则: 作用于同一数据表的多个任务及相应约束 q_i 和 l_i. 对于输出结果没有交集的类别进行去除.

在为单任务查询进行模型选择时, 需要保证将任务对应的类别约束完全覆盖. 而面对多任务查询时, 需要考虑不同任务及约束之间产生的影响. 在考量这个问题之前, 先从数据收集的层面上进行分析. 每个任务都有相应的备选模型库. 实际查询前, 模型库里的模型在待查询数据上的准确度是无从得知的. 因此每个模型都需要在历史数据中进行运行, 并获得模型在相应数据集上的准确率作为真实准确率的估计值. 实际上历史数据能够获取的也不只是模型的准确率. 在获取足够多的历史数据后, 可以对不同任务不同类别的数据相关性进行计算.

假定用户查询中存在 A 和 B 两个任务. 任务 A 和 B 对应的类别约束分别是 L_A 和 L_B, 且假定任务 A 上的分类比任务 B 上的分类成本更低. 考虑 L_B 中的一个子集 l_B. 如果经过任务 A 和 B 的分类之后, 从属于 L_A 的数据与从属于 l_B 的数据重合度低于某一阈值, 那么由于两个任务是合取的关系, 可以在任务 B 中省略 l_B 的部分而不会影响到整个查询的准确性. 从而任务 B 的类别约束从 L_B 缩小为 $L_B - l_B$. 在对任务 B 进行模型选择的时候就可以选择更少的模型来完成任务.

在新闻数据库中文本分类和实体分类的合取例子中, 实体分类任务相比文本分类任务更加困难. 因此缩减实体分类部分的成本更有利于降低查询的整体成本. 就文本分类任务的类别 (‘财经’, ‘金融’, ‘娱乐’, ‘家居’) 来讲, 当新闻正文从属于这些类别时, 正文对应标题为书名类别的可能性非常小. 由于这两个任务在查询中是合取的关系, 因此当文本分类任务完成之后, 它筛选的数据几乎没有书名类别. 从而在实体分类任务中进行模型选择时就可以减少诸如书名等类别.

在两个任务的基础上, 可以考虑多个任务的情况. 首先利用单任务查询优化中的模型选择算法, 获得每个任务对应的模型组合以及预期的计算成本. 基于预期成本, 可以把任务从低到高排列为 $Q = \{q_1,\ldots, q_m\}$. 任务对应类别约束为 $L_1,\ldots, L_m$. 然后考察每一对任务 Q_i 和 Q_j $(i < j)$ 相应的类别约束的重叠度. 由于排列在前的任务 Q_i 计算成本较低, 所以 Q_i 需要尽量处理更多的数据量. 当 L_j 中的类别 l 对应的数据与 L_i 的重叠度在给定的阈值以下时, 意味着 l 在任务 Q_j 中可以省略. 最后得到缩减版的类别约束.

基于模型选择的优化规则中, 优化收益来源于每个分类任务所需的模型数量的减少. 记各分类任务 $q_1,\ldots, q_k$ 对应的类别数量分别为 $l_1,\ldots, l_k$, 对应的估计成本分别为 $c_1,\ldots, c_k$, 两个任务类别之间数据的平均重叠度为 p, 则优化收益为 $profit_4 = p\sum_i \frac{l_i}{l} c_i$. 对于优化成本, 每次计算两个任务类别之间数据的重叠度的平均成本为 c'. 则优化成本为 $cost_4 = kc'\sum_i |l_i|$. 比较 $profit_4$ 和 $cost_4$ 来判断是否使用这一规则.

6.2 基于执行方式的优化

多分类任务下基于执行方式的优化规则: 将合取范式中的任务分成多个段, 段内任务的模型并行执行, 段间任务的模型串行执行.

考虑任务之间的串行/并行差别以及串行下任务执行的先后顺序. 以下述查询为例:

SELECT agency FROM news WHERE text_classifier(news.text, 90%) IN (‘财经’, ‘金融’, ‘科技’, ‘教育’, ‘娱乐’, ‘家居’) AND entity_classifier(news.title, 80%) IN (‘组织’, ‘地址’, ‘姓名’, ‘书名’);

当分类任务并行执行时, 任务中难度更高的实体分类任务就需要处理新闻表中的所有标题. 这导致查询时间变得非常长. 而如果先执行文本分类, 再执行实体分类的话, 就可以只保留 (‘财经’, ‘金融’, ‘科技’, ‘教育’, ‘娱乐’, ‘家居’) 这几个类别的数据. 从而后续实体分类任务的计算成本也会降低.

考虑任务组合 $Q = \{q_1,\ldots, q_k\}$. 任务的选择性基数 $S = \{s_1,\ldots, s_k\}$代表了任务对应的谓词的筛选程度. 即数据量为 B 的输入数据在经过任务 q_i 的计算和筛选后得到的数据量为 b_i, 有 $s_i = b_i/B$. 对应于数据量 B, 任务对应的模型组合有各自的计算成本 $C = \{c_1,\ldots, c_k\}$. 经过前面串行任务筛选后, 数据量会变少. 此时后续任务的计算成本也会

相应变少. 在介绍优化方法之前, 先引入一个定理.

将不同任务按照成本从小到大的顺序进行排列, 其中成本计算如下所示:

$$\frac{c_i}{1-s_i} \tag{16}$$

此时任务串行执行的估计计算成本为最小值, 如公式 (17) 所示:

$$\sum_{i=1}^{k}\left(\prod_{j=1}^{i-1}s_j\right)c_i \tag{17}$$

定理的证明如下.

用归纳法. 定义 $T(i, start, end)$ 为 q_i 最先执行时从 q_{start} 到 q_{end} 这个任务序列的最小计算时间. 首先证明起始条件, 即只有两个任务时定理是正确的, 也就是先执行 q_1 比 q_2 成本更低:

$$\begin{aligned}T(1,1,2)&=c_1+s_1c_2=c_2+s_2c_1+(1-s_2)c_1-(1-s_1)c_2\\&=c_2+s_2c_1+(1-s_1)(1-s_2)\left(\frac{c_1}{1-s_1}-\frac{c_2}{1-s_2}\right)\leqslant c_2+s_2c_1=T(2,1,2)\end{aligned} \tag{18}$$

接着, 当本定理对于 $2\leqslant k\leqslant n$ 均可满足时, 也就是说 $1\leqslant k\leqslant n$ 时, $T(1,1,k)=\min_{i=1,\ldots,k}T(i,1,k)$. 需要证明对于 $k=n+1$, 定理是可满足的. 对于 $k=n+1$, 考虑将除第 1 个任务以外的任务 q_u 放到第 1 的位置时的情况, 这将分为两种情况. 当 $u<n+1$ 时计算成本如下所示:

$$T(u,1,n+1)=c_u+s_uT(1,1,u-1)+\left(\prod_{j=1}^{u}s_j\right)T(u+1,u+1,n+1) \tag{19}$$

则有:

$$\begin{aligned}T(1,1,n+1)&=\sum_{i=1}^{n+1}\left(\prod_{j=1}^{i-1}s_j\right)c_i=T(1,1,u)+\left(\prod_{j=1}^{u}s_j\right)T(u+1,u+1,n+1)\\&\leqslant c_u+s_uT(1,1,u-1)+\left(\prod_{j=1}^{u}s_j\right)T(u+1,u+1,n+1)=T(u,1,n+1)\end{aligned} \tag{20}$$

分别把正确顺序版 $1\sim n+1$ 的任务序列和调换顺序版 u, $1\sim u-1$, $u+1\sim n+1$ 的任务序列都分成两部分. 各自的第 2 部分是从 q_{u+1} 到 q_{n+1} 的模型序列. 两个序列中这部分都是一样的. 各自的第 1 部分是两个不同的序列, 一个是从 q_1 到 q_u 的序列, 另一个选择把 q_u 移到前面. 由于前面的归纳条件, 正确顺序版任务序列的第 1 部分计算成本更低, 综合可知在 $u<n+1$ 时正确顺序版是成本最低的.

当 $u=n+1$ 时, 同样将 q_u 前移, 可得:

$$T(n+1,1,n+1)=c_{n+1}+s_{n+1}T(1,1,n-1)+\left(\prod_{j=1}^{n-1}s_j\right)s_{n+1}c_n \tag{21}$$

则有:

$$\begin{aligned}T(1,1,n+1)&=T(1,1,n-1)+\left(\prod_{j=1}^{n-1}s_j\right)(c_n+s_nc_{n+1})\leqslant T(1,1,n-1)+\left(\prod_{j=1}^{n-1}s_j\right)(c_{n+1}+s_{n+1}c_n)\\&=T(1,1,n-1)+\left(\prod_{j=1}^{n-1}s_j\right)c_{n+1}+\left(\prod_{j=1}^{n-1}s_j\right)s_{n+1}c_n\\&\leqslant c_{n+1}+s_{n+1}T(1,1,n-1)+\left(\prod_{j=1}^{n-1}s_j\right)s_{n+1}c_n)=T(n+1,1,n+1)\end{aligned} \tag{22}$$

综上所述, 在 $n\in Z^+$ 时, 任务序列 $q_1,\ldots,q_n$ 是计算成本最低的.

算法 2. 多任务确定执行方式的算法.

Input: 任务序列 $Q=\{q_1,\ldots,q_n\}$, 输入数据量 B;

Output: 任务的分段方式 $Seg(B, 1, n)$.

1. $min\_cost=\max_{1\leqslant i\leqslant n}Cost(q_i, B)$
2. $target=Q$
3. **for** $i=1,\ldots,n-1$ **do**
4. 计算 $Seg(B, 1, i)$ 并获取成本 *cost_left* 和筛选率 s

5. $B' = B \times s$
6. 计算 $Seg(B', i+1, n)$ 并获取成本 $cost\_right$
7. **if** $cost\_left + cost\_right < min\_cost$ **then**
8. $target = Seg(B, 1, i) \cup Seg(B', i+1, n)$
9. $min\_cost = cost\_left + cost\_right$
10. **return** $target$

依照这一定理, 调整任务之间的执行方式和顺序的安排, 如算法 2 所示. 首先将不同任务按照 $\frac{c_i}{1-s_i}$ 的值从低到高排序. 在此基础上, 逐步提高任务的并行度. 从所有任务并行执行的初始状态开始, 记录此时的总成本, 即所有任务中成本最高的:

$$cost = \max_{i=1,\ldots,n} Cost(q_i) \tag{23}$$

递归地对任务序列进行切分. 对已排序的任务序列 $q_1,\ldots, q_n$, 在算法 2 第 3–9 行的循环中, 将任务序列中每一个模型作为分界点, 将任务序列分成上半段和下半段两段. 第 4 行对上半段利用同一算法获得它的任务切分、总的计算成本以及数据的筛选比率. 其中筛选比率可以被用来计算第 5 行下半段的输入数据量. 基于计算出来的输入数据量, 在第 6 行中对下半段同样计算获得任务切分、计算成本. 两端的计算成本之和即为分界后的总成本. 由于我们维护了全局的最小成本, 因此在第 7 行中将分段后的总成本与维护的全局最小总成本比较以判断是否进行切分 (第 8 行), 同时更新全局最小总成本 (第 9 行). 循环结束后将成本最低的分段方式作为结果返回.

为了证明分段的最优性, 使用归纳法. 第 1 步, 在任务数量为 2 的时候, 上述算法会比较两个任务并行和串行的成本以选出较优的方法. 第 2 步, 在任务数量 k 满足 $2 \leqslant k \leqslant n$ 时算法可以获得最优解的前提下, 需要证明 $k = n+1$ 时算法可以获得最优解. 假定算法计算的最优分段方式为 A, 同时存在另一种分段 B 比 A 更优. 由于两种分段方式至少存在一个分界点不同, 分别记为 p_A, p_B. 由于 $2 \leqslant k \leqslant n$ 时算法可以获得最优解, 对于分段方式 B 来说 p_B 前后两端均为使用算法分段得到的, 否则使用算法可以获得比分段方式 B 更优的分段. 那么根据算法, p_B 是比 p_A 更差的分界点, 这与假设不相符, 得证分段算法的最优性.

基于执行方式的优化规则中, 在不采用该规则的情况下默认使用并行执行方式, 其估计计算成本为 $\max_i c_i$. 使用规则后, 记每个任务的平均计算成本为 $\overline{c_q}$, 平均筛选率为 $\bar{s}$, 历史数据中统计的平均每个段内模型数量为 x , 则预估计算成本为 $\bar{c}\frac{1-(1-\bar{s})^k}{1-(1-\bar{s})^x}$, 优化收益为 $profit_5 = \max_i c_i - \bar{c}\frac{1-(1-\bar{s})^k}{1-(1-\bar{s})^x}$. 对于优化成本, 由于使用了递归的方法计算分段, 记计算每种分段的成本为 c_s, 则优化成本为 $cost_5=2^k c_s$. 比较 $profit_5$ 和 $cost_5$ 来判断是否使用这一规则.

6.3 基于设备绑定的优化

多分类任务下基于设备绑定的优化规则: 任务段内的模型逐一绑定低成本设备.

经过第 6.2 节的计算, 模型被分成 k 个段. 每个段内的模型在并行执行后输出结果. 将运行结果进行合并之后便可以对数据进行筛选. 段与段之间的模型将会顺序执行. 前面的模型执行之后会在其输入数据量的基础上筛选数据. 从而每个段内模型接受的数据量为 $b_1>b_2>\ldots>b_k$. 后续模型的计算成本因数据量的降低而降低.

在考虑设备绑定时, 先对每个段内的模型进行考虑: 由于每个段内的模型可以并行执行, 因此将模型绑定到尽可能多的设备. 通过充分利用设备以减少段内模型推断的总时间. 这部分的计算与单任务并行模型组合的设备绑定的计算相同. 使用单任务中基于设备绑定的方法进行同一个模型段内的设备绑定. 对于段间的设备绑定: 由于前一段的模型全部计算完成之后才会运行后一段的模型. 因此段间的设备绑定互不影响. 最后将每个段的设备绑定综合起来, 得到整个查询计划的设备绑定.

7 实验分析

本文对提出的优化规则进行了详细的实验, 并将本文方法与 UDF 实现模型、RETRO[16]、SQL 实现模型[1]、概率谓词[2]、相关代理模型[20]、任务无关索引[21]等方式进行比较. 所有实验都在一台使用 64 位 Ubuntu 操作系

统、内核版本为 4.4.0-210-generic 的服务器上进行, 内存为 187 GB, 处理器为 Intel(R) Xeon(R) Silver 4110 CPU@2.10 GHz, GPU 为 NVIDIA Quadro P5000. 实验是在列式存储关系型数据库 MonetDB 上进行的. 使用 C/C++实现优化规则. 在实现机器学习模型的调用时, 使用 Libtorch 模块将训练完毕的 PyTorch 模型转换成 C/C++形式. 实验中使用 4 个数据集分别是 THUCNews[22], CLUENER[23], CIFAR-10 和 CIFAR-100[24]. 将这些数据集中的输入数据作为非结构化数据与关系表进行结合以供查询.

实验的评价指标如下所示.

(1) 分类精度. 本文主要使用准确率 P 和召回率 R 作为分类精度指标. 给定类别 c 和数据量为 N 的数据集 D, 数据中类别 c 的数量为 N_c. 分类为 c 的数据量为 N_p, 其中实际为类别 c 的数量为 N_{pc}. 从而准确率 p_c 和召回率 r_c 如下所示:

$$p_c = \frac{N_{pc}}{N_p} \tag{24}$$

$$r_c = \frac{N_{pc}}{N_c} \tag{25}$$

整体的准确率 P 和召回率 R 如下所示:

$$P = \sum\nolimits_c \frac{N_p p_c}{N} \tag{26}$$

$$R = \sum\nolimits_c \frac{N_c r_c}{N} \tag{27}$$

(2) 查询时间. 给定查询 Q、数据集 D 和设备 dev, 执行查询的实际时间如下所示 (单位: s):

$$t^{Q}_{D,dev} \tag{28}$$

实验对比方法有以下几个.

(1) 基于 UDF 的协同查询: 没有优化的方法.

(2) RETRO[16]: 针对拥有文本列的数据库的研究工作. 由于许多文本上的机器学习模型需要利用文本的向量表示进行计算, 因此该研究为这些文本列中的每个单词提前赋予预训练的词向量, 从而避免了在每次调用模型之前对文本进行转换的过程.

(3) SQL 实现[1]: 在数据库中使用卷积神经网络进行图像数据推断的方法. 他们将神经网络的模型参数和输入数据都存储为关系表的形式, 并对输入数据的每个值, 按照卷积操作中所在的块位置和块内位置的索引进行存储. 这样, 他们可以使用 SQL 来实现卷积和池化的操作, 从而在数据库中实现卷积神经网络的推断过程.

(4) 概率谓词[2]: 基于一类特殊的谓词. 这类谓词中查询者要求某一个类别的数据, 如“$f(T.c)$= ‘Type-a’”这样的二元分类谓词. 针对这一任务, 可以训练一个只负责判断数据点是否属于‘Type-a’类别的二分类代理模型. 该代理模型在体量和推断时间上都要远低于应用于所有类别的分类模型, 即参考模型, 从而大大降低查询的反馈时间.

(5) 相关代理模型[20]: 同一个查询中可能涉及多个包含分类任务的判断条件, 且这些条件之间存在相关关系. 当两个判断条件的输出结果是独立的时候, 使用两个独立的代理模型进行推断可能会浪费一些推断成本. 因此, 他们提出了在线训练代理模型, 并考虑它们的组合方式, 分配它们的精度参数, 并将它们放入修改后的查询计划中. 为了减少在输入查询被加速之前训练代理模型的计算开销和延迟, 查询优化器会重复使用中间结果, 并使用分支和边界搜索来修剪候选查询计划.

(6) 任务无关索引[21]: 引入特殊的索引来实现的. 该索引由一个体量较小的嵌入网络和相似度指标构成. 在查询之前, 所有数据点都会经过此索引, 被转换成嵌入向量. 基于这些嵌入向量的相似度指标可以进行聚类, 从每个聚类中挑选出一些具有代表性的数据, 用复杂网络计算结果作为聚类核心. 因此, 在查询时, 根据查询谓词的不同, 可以从相应的聚类中选择数据, 用复杂网络计算结果.

7.1 查询优化的整体效果

为了测试本文提出的优化规则的性能, 将使用以下查询在不同方法上进行评估.

查询 1: SELECT $f(T.c)$ FROM T;

查询 2: SELECT * FROM T WHERE *f*(*T.c*, 90%) ='A';

查询 3: SELECT * FROM T WHERE *f*(*T.c*, 90%) IN ('A', 'B', 'C', 'D');

查询 4: SELECT * FROM Ta, Tb WHERE *f*(Ta.c, 90%) IN ('A', 'B', 'C', 'D') AND g(Tb.d, 95%) IN ('E', 'F', 'G', 'H');

查询 5: SELECT * FROM Ta JOIN Tb ON Ta.a=Tb.b WHERE *f*(Ta.c, 90%) IN ('A', 'B', 'C', 'D');

查询 6: SELECT * FROM T WHERE *f*(*T.c*, 90%) IN ('A', 'B', 'C', 'D') GROUP BY *T.d*;

查询的反馈时间如表 1 所示, 其中第 1 列为对比的各方法, 使用指标为查询时间, 单位为秒 (s). 从查询 1 的时间比较可以看出, RETRO、SQL 实现等与查询类别无关的方法可以起到优化作用. 相比基于 UDF 的协同查询的方式, 查询时间都有不同程度的降低. 可以看出 SQL 实现方式在这一查询中的优化效果最好. 而对其他的优化方法而言, 这一类查询没有提供约束信息. 从而这些方法无法利用查询中对 DIF 输出结果的约束进行优化. 本文的优化方法也需要查询提供的信息, 所以同样无法起到优化效果.

表 1 查询的优化性能对比 (s)

优化方式	查询1	查询2	查询3	查询4	查询5	查询6
基于UDF的协同查询	82.1	85.4	85.2	114.5	92.8	88.2
RETRO	68.8	71.1	74	97.5	78.7	77.2
SQL实现	**43.7**	44.2	43.9	58.1	55.3	56.8
概率谓词	82.2	**6.5**	16.8	24.6	16.9	18.8
相关代理模型	82.3	7.3	18.1	20.2	17.6	19.4
任务无关索引	82.5	8.4	34.1	52.5	35.9	38
本文优化方法	82.2	8.9	**10.3**	**15**	**8.6**	**9.4**

查询 2–查询 6 为基于声明式推理的协同查询. 相比于查询 1, 查询 2 在谓词中为任务提供单个类别的约束. 在查询 2 的实验结果中, RETRO 和 SQL 实现等方法仍可以降低查询时间. 但由于无法利用到查询给定的约束信息, 其优化效果有限. 其他的优化方法可以对提供的约束信息进行使用, 相比之下有更大的优化空间. 针对提供的单个类别, 概率谓词方法可以快速训练体量很小的模型, 如线性二分类模型等, 因此它的查询时间是实验方法中最短的. 本文优化方法作用下的查询时间比概率谓词方法稍长. 分析查询优化器实际选择的模型, 发现模型覆盖了包括目标类别的多个类别. 由于模型体量更大, 其计算成本更高. 但与 RETRO 和 SQL 实现等方法相比, 仍旧有不小的提升.

在查询 3 中提供了多类别的约束. 概率谓词和相关代理模型等方法需要对每一个类别都训练一个代理模型. 这一做法引入了更多的训练时间. 同时查询执行时需要运行每一个代理模型, 这也增加了查询时间. 从而这些方法的查询时间大幅延长. 而本文的优化方法会在已有的模型库中挑选数量较少、模型体量也比较小的组合. 在节省训练时间的同时, 模型执行的时间也比其他代理模型的方法的模型组合更短, 从而实现更加高效的协同查询.

查询 2 和查询 3 是单任务的情形, 而查询 4 考虑了多个任务的情况. 对关系表中的数据列执行不同的分类任务时, 任务的输出结果之间可能存在一定的相关性. 为了测试存在相关性时的优化效果, 本文对数据列、分类任务和类别约束的选取进行调节, 使查询 4 中不同分类任务的筛选结果有不相交部分. 在概率谓词方法与相关代理模型的对比中可以发现, 后者对于多任务的查询有针对性优化, 从而实现更短的查询时间. 本文的优化同样考虑到多任务的因素. 相较于相关代理模型方法, 使用的模型更少, 进一步降低了查询时间.

针对其他算子如 JOIN、GROUP BY 等存在的情况, 添加查询 5 和查询 6 进行测试. 在 SELECTION 算子与其他算子共同使用之后, 本文的优化规则仍可以有效降低查询成本.

7.2 查询精度的影响

在第 7.1 节的实验中, 在对给定的查询进行测试时都固定了每个任务所选取的精度指标和阈值. 本节将在同一任务里选择不同的精度阈值, 以测试查询精度的影响. 由于精度对不同难度的任务的影响程度不同, 因此本节实验会区分简单任务 (如 THUCNews) 和复杂任务 (如 CLUENER) 来进行. 两种任务对应的类别集合分别为{体育, 财经, 娱乐, 游戏, 家居, 房产}和{组织, 人名, 位置, 场景}. 此处简单任务和复杂任务并非特指某些任务, 二者主要

体现在任务的计算时间的差别. 给定精度阈值 p, 任务 q_1, q_2 在同一数据量 D 下完成分类并且精度达到 p 所需的时间分别为 t_1, t_2. 当 $t_1 << t_2$ 时, 我们称 q_1 为简单任务, 而 q_2 为复杂任务. 例如, 由实际任务计算时间测试可知, THUCNews 数据集上的分类任务即为简单任务, 而 CLUENER 数据集上的分类任务为复杂任务.

本实验中, 设置两种任务对应的类别集合分别如下.

简单任务: {体育, 财经, 娱乐, 游戏, 家居, 房产}; 复杂任务: {组织, 人名, 位置, 场景}. 图 4 和图 5 分别给出了查询精度对两种任务影响的实验结果.

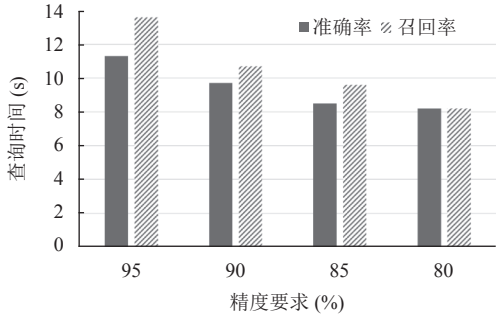


图 4 简单任务查询在不同精度要求下的查询时间

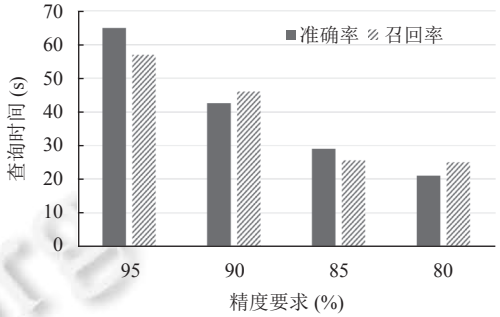


图 5 复杂任务查询在不同精度要求下的查询时间

对复杂任务来说, 降低精度阈值可以让优化器有更多的模型组合可以选择, 因此查询时间会有明显的降低. 与之相反, 对简单任务来说, 由于使用一些较为简单的模型往往就有比较高的精度, 所以再继续降低精度阈值对性能的提升作用不大.

7.3 模型选择的影响

对于模型选择这一角度为优化带来的影响, 分成单任务和多任务两种情况进行实验和评估.

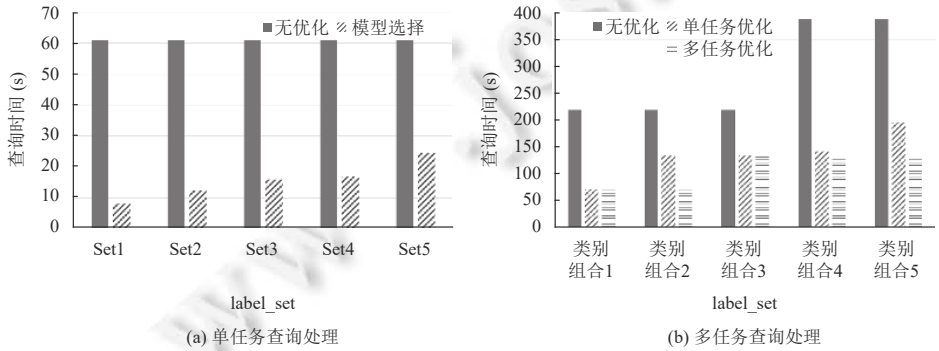
对于单任务的环境下, 使用以下查询来进行测试:

SELECT * FROM T WHERE function(*T.c*, 90%) IN label_set;

将 THUCNews、CIFAR-10 等数据集的输入数据作为数据列 *T.c*. 对于模型选择的优化规则来讲, label_set 就是影响查询任务找到的模型组合的主要因素. 探究对于同一个任务来说, 取值不同、大小不同的 label_set 会如何影响查询的反馈时间.

为了更好地体现不同 label_set 带来的影响, 在优化器中只使用模型选择相关的规则, 并且对于其他影响因素如设备等也进行限制, 尽量保证提供统一的运行环境. 将模型放在 CPU 上执行, 并通过限制 CPU 的使用率在同样的水平来保证环境相同.

在 THUCNews 上的实验结果如图 6(a) 所示, 采用 5 组 label_set, 分别是: Set1={体育, 财经}, Set2={家居, 房产}, Set3={游戏, 科技}, Set4={体育, 财经, 娱乐, 游戏}和 Set5={体育, 财经, 娱乐, 游戏, 教育, 科技}.



(a) 单任务查询处理 (b) 多任务查询处理

图 6 查询处理中模型选择的影响

首先是使用模型选择的前后对比. 在图 6(a) 中对比每一组的运行时间可以发现, 单独使用模型选择规则可以有效地降低查询处理过程中的成本, 表明这条规则本身在优化中是有效的.

其次是不同类别数量下模型选择的影响对比. 在第 1 组、第 4 组和第 5 组的对比中, 通过逐步添加类别数量, 并考察他们的执行时间差别, 可以知道, 随着类别数量的增加, 查询获取结果的时间也在逐渐提高. 查询其实际实现方式可知, 类别的增加通常在优化器中会导致选择更多的模型, 或者用更加复杂、可以覆盖更多类别但耗时更长的模型来实现.

最后是相同类别数量下不同模型选择策略的影响对比. 将第 1 组和第 2 组进行对比, 这两组 label_set 的类别数量是一致的, 只是实际的类别不同. 从图 6(a) 中可以看出, 第 2 组的查询反馈时间明显长于第 1 组. 查看查询日志可以发现, 对这两组类别, 优化器选择的模型体量有所不同. 第 2 组的模型体量更大, 需要更多的计算. 而在第 1 组和第 3 组的对比中, 尽管处理的类别数量相同, 但计算时间还是有明显差别. 在查询日志中发现第 3 组与第 1 组的差别在于, 第 3 组中优化器选用了更多的模型, 从而在模型推理上花费更多的时间.

对多任务的模型选择的评估, 除了每个任务自身使用单任务的模型选择算法选出适合自己的模型组合外, 还需要考量不同任务之间的相互影响. 使用以下的查询来考察多任务的模型选择的影响:

SELECT * FROM T WHERE function1(*T.c*, 90%) IN label_set1 AND function2(*T.d*, 90%) IN label_set2;

上述查询中, *T.c* 和 *T.d* 是来自同一关系表的两个不同的非结构化数据列, 分别对应于两个不同的机器学习数据集. function1, function2 代表着对应于 *T.c* 和 *T.d* 的机器学习任务. 而与单任务类似, label_set1 和 label_set2 是影响模型选择的重要因素. 此处评估对于不同任务, 调节 label_set 会产生的影响.

在实验之前必须声明, 由于两个数据集的分布相关性是未知的, 本实验在将两个数据集放到关系表中合成需要的关系表时, 实际上是可以控制不同类别的对应关系的. 比如让其中一个数据集里的某一个类别的数据, 对齐到另一个数据集的部分类别数据. 这样的有较强相关关系的数据分布实际上就很适合多任务模型选择的规则发挥. 也可尝试将数据集中的数据随机分布到关系表中, 这种情况下任务之间的相关性非常低, 缺乏让优化器进行优化的条件. 在下文的实验中会使用第 1 种方式, 也就是合成数据时考虑任务间进行数据对齐.

对其中一组设置进行实验, 即: *T.c* 和 *T.d* 分别选取来自 THUCNews 和 CLUENER 的数据, 这两组数据所对应的任务具备不同的计算难度, 前者的模型所需要的计算时间普遍要低于后者. 并且也选出几种不同种类的 label_set1 和 label_set2 组合.

类别组合 1: {体育, 财经}和{组织, 人名}

类别组合 2: {体育, 财经}和{组织, 人名, 书籍, 政府}

类别组合 3: {体育, 财经}和{组织, 人名, 位置, 场景}

类别组合 4: {体育, 财经, 娱乐, 游戏}和{组织, 人名, 书籍, 政府}

类别组合 5: {体育, 财经, 娱乐, 游戏}和{组织, 人名, 地址, 公司, 政府, 书籍}

实验结果如图 6(b) 所示, 横坐标的类别组合对应于上述的组合.

首先是使用模型选择前后对比. 对比每一组 label_set 组合内的数据, 只使用单任务的模型选择规则时, 也可以较大程度地降低查询所需的时间, 而多任务的规则相比于单任务的优化又在不同的场景下有不同程度的优化.

其次是不同类别数量的对比. 对比第 1 组和第 2 组可以发现, 在 THUCNews 对应的任务类别约束不变的情况下, 增加了 CLUENER 对应的类别约束时, 单任务的优化规则会选择更多的、或者更复杂的模型来应对更多的类别约束. 而对比第 2 组和第 3 组, 当 CLUENER 新添加的类别与 THUCNews 的类别约束存在较强的相关性时, 多任务的模型选择规则就无法使用, 从而比起有优化的查询计算成本也会更高. 从第 2 组和第 4 组的对比结果来看, 当 THUCNews 的类别约束增加时, 对应的结果数据增加, 可能会覆盖之前没有相关性的 CLUENER 的类别, 从而 CLUENER 之前可以被多任务的规则省略掉的类别也变得不可忽略, 从而无法起到降低成本的作用.

7.4 执行方式的影响

为了考察执行方式为优化带来的影响, 分成单任务和多任务两种情况进行实验和评估. 在本节的实验中, 基于模型选择的优化规则, 为每个选择的任务和对应类别约束挑选相应的模型集合, 以便于测试执行方式的规则.

对于单任务的环境下, 使用与第 7.3 节实验相同的查询. 查询中 label_set 的大小以及里面具体的类别信息会影响到模型选择时选出来的模型组合的大小以及具体模型的计算成本, 而这同时也会对执行方式的选择带来不同的影响. 而在查询之外, 不同设备也会让优化器倾向于不同的执行方式, 所以设备的影响也需要纳入考虑范畴. 具体来说, 使用 THUCNews 数据集的文本作为 *T.c* 的内容, 在 label_set 的选择上选用了 Set1={体育, 财经}, Set4={体育, 财经, 娱乐, 游戏}, Set5={体育, 财经, 娱乐, 游戏, 教育, 科技}和 Set6={体育, 财经, 娱乐, 游戏, 家居, 房产}这几组. 在设备上, 通过 CPUlimit 可以对 MonetDB 的 CPU 的利用率进行调整, 从而提供两个设备选项: 设备容量充足和受限.

实验结果如图 7 所示, 从组间的比较可以看出, 随着选择的模型数量增加 (第 1、2、3 组) 或者模型的复杂程度增加 (第 3 组和第 4 组), 计算成本也在随之提升. 接着按照设备的容量是否充足分为两类进行讨论.

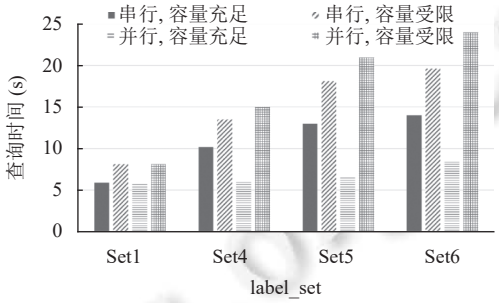


图 7 单任务执行方式选择的影响

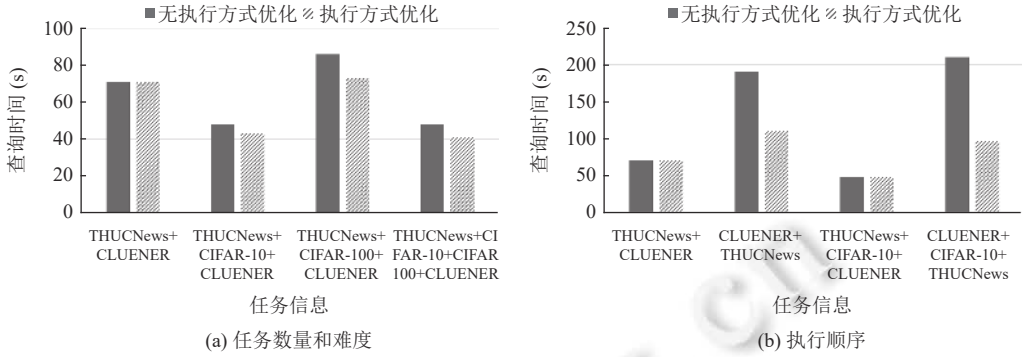
对于设备容量充足的, 串行和并行两种执行方式对应实验结果图的第 1 列和第 3 列. 从图 7 中来看, 随着模型数量和复杂程度的上升, 对于串行执行的方式来说, 它所需的计算成本也会随之稳步上升. 由于其中的每个模型都只覆盖很小部分的类别, 所以模型与模型的执行之间筛选数据的效率很低, 从而后续的模型所需要处理的数据并没有削减太多. 这种情况下设备资源没有被很好地使用. 相比之下, 并行执行的计算成本上升幅度很小, 因为它充分利用了计算设备, 让每个模型获得足够的计算资源以进行计算. 在这种情况下, 优化规则会根据估计出来的成本选择并行的执行方式.

而对于设备容量受限的, 包括存储容量和计算容量的情况, 串行和并行两种执行方式对应实验结果图的第 2 列和第 4 列. 首先相比于容量充足的情况, 容量受限时, 无论是串行和并行都受到一定程度的影响, 提高了计算成本. 而就并行执行来说, 由于多个模型竞争计算资源, 最终呈现出计算成本随模型增加的提升幅度较大的情况. 相反, 串行执行之下, 每个模型都可以单独使用有限的计算资源, 并且输出筛选的机制也让后续的模型可以减少处理的数据量, 从而对每一组 label_set 来说, 串行的计算成本都要略低于并行执行.

对于多任务的情形, 任务的数量、任务的计算难度差异和类别约束的不同等都可能影响到优化规则的效果, 本实验也会测试不同的影响因素与优化规则效果之间的关系. 使用与第 7.3 节相同的查询. 任务的选择上, 选择 THUCNews 和 CIFAR-10 作为简单任务的代表, 而 CLUENER 和 CIFAR-100 作为复杂任务的代表. 在实验之前需要阐明使用与不使用多任务执行方式优化规则的区别, 在 MonetDB 中, 如果有多个包含机器学习任务的谓词进行合取, 那么: 第一是这些谓词会串行执行, 前面的任务执行完之后会对数据进行筛选. 第二是执行的顺序, MonetDB 会按照选择度基数 (selectivity cardinality) 的大小对谓词进行排序, 尽量让筛选度更高的谓词先执行以减少后续需要处理的数据. 以上是不使用规则时的默认方式, 而本文的优化规则所选择的方式不一定与默认选择的方式不同, 适用的场景也有部分限制.

首先是任务的数量和难度配置, 如图 8(a) 所示, 横坐标代表着使用到的任务信息. 如 THUCNews+CLUENER 代表查询中包含 THUCNews 和 CLUENER 两个数据集对应的任务. 随着任务的增多, 计算成本并不一定会随之增加, 这是因为前面任务的筛选过程减少了后续任务的计算成本. 因此, 将主要比较优化规则对于成本的影响. 第 1 组中优化规则选取的方案与默认的相同, 所以并没有产生区别. 在第 2 组和第 3 组的对比中, 第 3 组的成本削减比第 2 组更大. 这是因为第 2 组有两个简单任务, 而第 3 组有两个复杂任务. 相比于默认的串行执行方式, 优化规则让难度相近的任务并行执行, 而难度相差较大的串行执行. 所以第 3 组在让复杂任务并行执行所节省的时间比第

2 组更多. 而对于第 4 组, 由于前面的任务比较简单且筛选了更多的数据, 后续复杂任务需要处理的数据更少, 所以优化规则降低的计算成本少一些.



(a) 任务数量和难度 (b) 执行顺序

图 8 多任务执行方式选择的影响

为了测试执行顺序的影响, 首先限制设备的算力来促使任务顺序执行, 然后我们会通过调节简单任务 (如 THUCNews) 和复杂任务 (如 CLUENER) 的类别约束大小比例来进行评估. 如图 8(b) 所示, 横坐标代表着使用到的任务信息. 如 THUCNews+CLUENER 代表查询中 THUCNews 和 CLUENER 两个数据集对应任务, 并且在查询中是按照 THUCNews→CLUENER 的顺序出现的. 在第 1 组和第 2 组的对比中, 使用 THUCNews 和 CLUENER, 第 1 组给 THUCNews 更少的类别约束, 第 2 组反之. 结果显示, 第 1 组的执行顺序相同, 而第 2 组中无优化时复杂任务先执行, 需要耗费大量的时间, 而优化规则选择让简单任务先执行以筛选数据, 从而降低了时间成本. 第 3 组和第 4 组添加了一个任务, 同样按照任务的复杂性从简到繁以及从繁到简对类别约束进行了从少到多的分配, 结果也与前面类似. 第 3 组双方一致, 而第 4 组优化规则将任务的执行顺序进行调整, 有效地降低计算时间.

7.5 设备选择的影响

本节实验需要考虑设备的分配问题. 由于硬件条件的限制, 服务器只有 CPU 和 GPU 可以使用, 其他设备只能使用模拟的方式来提供. 并且相比于本实验提出的任务来说, 设备的容量 (无论是存储还是算力) 都是非常充足的, 所以我们要模拟有不同容量的设备时, 需要人为地对设备进行限制.

由于本文对设备选择的支持以每一组并行执行的模型为单位进行, 所以我们会选用在前面规则支持下选择并行执行模型的查询, 本节选择的是这个查询:

SELECT * FROM T WHERE function1(*T.c*, 90%) IN label_set1 AND function2(*T.d*, 90%) IN label_set2;

在该查询的 function1 和 function2 的选择方面, 使用 CIFAR-100 和 CLUENER, 并且扩大 label_set 使优化规则选择更多的模型, 从而在多设备时更能体现优化的效果. 在设备方面, 对 CPU, GPU 的容量和算力做限制, 一方面模拟更加实际的情况, 另一方面也可以模拟出一个新的设备 Dev, 它的算力在 CPU 和 GPU 之间. 实验结果如图 9 所示, 无优化时默认会将模型都部署到算力最强的设备上, 如 GPU. 但它的容量是有限的, 所以在第 2、3、4 组逐步引入其他设备后, 优化规则会将模型部署到不同设备上以充分利用计算资源, 降低时间成本.

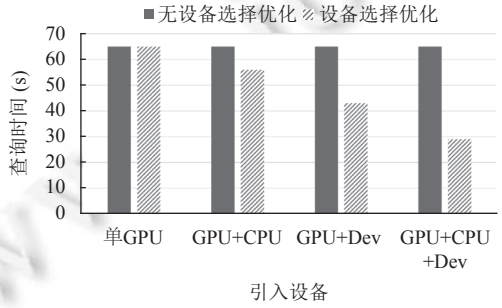


图 9 设备选择的影响

8 总结与展望

本文针对时间敏感场景中基于声明式推理的协同查询的性能问题, 提出了基于代理模型的优化规则. 对 DIF 对应的分类任务预先训练对应不同类别集合、精准度和计算时间的模型库. 利用模型库中的模型组合实现 DIF. 构建 DIF 实现方案选择的成本模型, 从而对用户查询中每个 DIF 应该选择的模型子集、模型所在的设备位置、执行顺序等进行考量, 并提供成本最低的实现方案, 从而最大幅度降低查询时间. 但本文工作仍存在以下不足之处: 对每一种机器学习任务都需要预先训练大量模型, 增加查询前的计算成本. 同时用在其他算子如聚合、连接、限制等上的相应优化还有所欠缺.

References:

[1] Lin QR, Wu S, Zhao JB, Dai J, Li FF, Chen G. A comparative study of in-database inference approaches. In: Proc. of the 38th IEEE Int'l Conf. on Data Engineering. Kuala Lumpur: IEEE, 2022. 1794–1807. [doi: 10.1109/ICDE53745.2022.00180]

[2] Lu Y, Chowdhery A, Kandula S, Chaudhuri S. Accelerating machine learning inference with probabilistic predicates. In: Proc. of the 2018 Int'l Conf. on Management of Data. Houston: ACM, 2018. 1493–1508. [doi: 10.1145/3183713.3183751]

[3] Li GL, Zhou XH, Sun J, Yu X, Yuan HT, Liu JB, Han Y. A survey of machine learning based database techniques. Chinese Journal of Computers, 2020, 43(11): 2019–2049 (in Chinese with English abstract). [doi: 10.11897/SP.J.1016.2020.02019]

[4] Li GL, Zhou XH. Survey of data management techniques for artificial intelligence. Ruan Jian Xue Bao/Journal of Software, 2021, 32(1): 21–40 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6121.htm [doi: 10.13328/j.cnki.jos.006121]

[5] Sun LM, Zhang SM, Ji T, Li CP, Chen H. Survey of data management techniques powered by artificial intelligence. Ruan Jian Xue Bao/Journal of Software, 2020, 31(3): 600–619 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5909.htm [doi: 10.13328/j.cnki.jos.005909]

[6] Chai MK, Fan J, Du XY. Learnable database systems: Challenges and opportunities. Ruan Jian Xue Bao/Journal of Software, 2020, 31(3): 806–830 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5908.htm [doi: 10.13328/j.cnki.jos.005908]

[7] Qiu T, Wang B, Shu ZW, Zhao ZB, Song ZW, Zhong YH. Intelligent index tuning approach for relational databases. Ruan Jian Xue Bao/Journal of Software, 2020, 31(3): 634–647 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5906.htm [doi: 10.13328/j.cnki.jos.005906]

[8] Li GL, Zhou XH. XuanYuan: An AI-native database systems. Ruan Jian Xue Bao/Journal of Software, 2020, 31(3): 831–844 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5899.htm [doi: 10.13328/j.cnki.jos.005899]

[9] Oracle. Oracle advanced analytics. 2012. https://www.oracle.com/artificial-intelligence/database-machine-learning/features/

[10] Microsoft. Microsoft SQL MLS. 2017. https://learn.microsoft.com/en-us/sql/machine-learning/?view=sql-server-2017

[11] Hellerstein JM, Ré C, Schoppmann F, Wang DZ, Fratkin E, Gorajek A, Ng KS, Welton C, Feng XX, Li K, Kumar A. The MADlib analytics library: Or MAD skills, the SQL. Proc. of the VLDB Endowment, 2012, 5(12): 1700–1711. [doi: 10.14778/2367502.2367510]

[12] D'Silva JV, de Moor F, Kemme B. AIDA: Abstraction for advanced in-database analytics. Proc. of the VLDB Endowment, 2018, 11(11): 1400–1413. [doi: 10.14778/3236187.3236194]

[13] Li XP, Cui B, Chen YR, Wu WT, Zhang C. MLog: Towards declarative in-database machine learning. Proc. of the VLDB Endowment, 2017, 10(12): 1933–1936. [doi: 10.14778/3137765.3137812]

[14] Luo SY, Gao ZJ, Gubanov M, Perez LL, Jermaine C. Scalable linear algebra on a relational database system. IEEE Trans. on Knowledge and Data Engineering, 2019, 31(7): 1224–1238. [doi: 10.1109/TKDE.2018.2827988]

[15] Schüle ME, Simonis F, Heyenbrock T, Kemper A, Günnemann S, Neumann T. In-database machine learning: Gradient descent and tensor algebra for main memory database systems. In: Proc. of Datenbanksysteme für Business, Technologie und Web (BTW 2019), 18. Fachtagung des GI-Fachbereichs, Datenbanken und Informationssysteme. Rostock: Gesellschaft für Informatik, 2019. 247–266.

[16] Günther M, Thiele M, Lehner W. RETRO: Relation retrofitting for in-database machine learning on textual data. In: Proc. of the 23rd Int'l Conf. on Extending Database Technology. Copenhagen: OpenProceedings.org, 2020. 411–414.

[17] Kang DL, Mathur A, Veeramacheneni T, Bailis P, Zaharia M. Jointly optimizing preprocessing and inference for DNN-based visual analytics. Proc. of the VLDB Endowment, 2020, 14(2): 87–100. [doi: 10.14778/3425879.3425881]

[18] Niu ZP, Li GL. In-database AI model optimization. Ruan Jian Xue Bao/Journal of Software, 2021, 32(3): 622–635 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6179.htm [doi: 10.13328/j.cnki.jos.006179]

[19] Kang DL, Emmons J, Abuzaid F, Bailis P, Zaharia M. NoScope: Optimizing neural network queries over video at scale. Proc. of the VLDB Endowment, 2017, 10(11): 1586–1597. [doi: 10.14778/3137628.3137664]

[20] Yang ZH, Wang ZZ, Huang YC, Lu Y, Li C, Wang XS. Optimizing machine learning inference queries with correlative proxy models. Proc. of the VLDB Endowment, 2022, 15(10): 2032–2044. [doi: 10.14778/3547305.3547310]

[21] Kang DL, Guibas J, Bailis P, Hashimoto T, Zaharia M. Task-agnostic indexes for deep learning-based queries over unstructured data. arXiv:2009.04540, 2020.

[22] Li JY, Sun MS, Zhang X. A comparison and semi-quantitative analysis of words and character-bigrams as features in Chinese text categorization. In: Proc. of the 21st Int'l Conf. on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics. Sydney: Association for Computational Linguistics, 2006. 545–552. [doi: 10.3115/1220175.1220244]

[23] Xu L, Tong Y, Dong QQ, Liao YX, Yu C, Tian Y, Liu WT, Li L, Liu CQ, Zhang XW. CLUENER2020: Fine-grained named entity recognition dataset and benchmark for Chinese. arXiv:2001.04351. 2020.

[24] Krizhevsky A. Learning multiple layers of features from tiny images [MS. Thesis]. Toronto: University of Toronto, 2009.

附中文参考文献:

[3] 李国良, 周煊赫, 孙佶, 余翔, 袁海涛, 刘佳斌, 韩越. 基于机器学习的数据库技术综述. 计算机学报, 2020, 43(11): 2019–2049. [doi: 10.11897/SP.J.1016.2020.02019]

[4] 李国良, 周煊赫. 面向 AI 的数据管理技术综述. 软件学报, 2021, 32(1): 21–40. http://www.jos.org.cn/1000-9825/6121.htm [doi: 10.13328/j.cnki.jos.006121]

[5] 孙路明, 张少敏, 姬涛, 李翠平, 陈红. 人工智能赋能的数据管理技术研究. 软件学报, 2020, 31(3): 600–619. http://www.jos.org.cn/1000-9825/5909.htm [doi: 10.13328/j.cnki.jos.005909]

[6] 柴茗珂, 范举, 杜小勇. 学习式数据库系统: 挑战与机遇. 软件学报, 2020, 31(3): 806–830. http://www.jos.org.cn/1000-9825/5908.htm [doi: 10.13328/j.cnki.jos.005908]

[7] 邱涛, 王斌, 舒昭维, 赵智博, 宋子文, 钟延辉. 面向关系数据库的智能索引调优方法. 软件学报, 2020, 31(3): 634–647. http://www.jos.org.cn/1000-9825/5906.htm [doi: 10.13328/j.cnki.jos.005906]

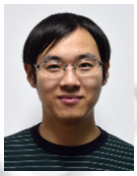
[8] 李国良, 周煊赫. 轩辕: AI 原生数据库系统. 软件学报, 2020, 31(3): 831–844. http://www.jos.org.cn/1000-9825/5899.htm [doi: 10.13328/j.cnki.jos.005899]

[18] 钮泽平, 李国良. 数据库内 AI 模型优化. 软件学报, 2021, 32(3): 622–635. http://www.jos.org.cn/1000-9825/6179.htm [doi: 10.13328/j.cnki.jos.006179]

邱志林(1997－), 男, 硕士, 主要研究领域为数据库内机器学习的优化.

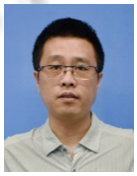
江大伟(1982－), 男, 博士, 研究员, 博士生导师, 主要研究领域为分布式数据管理技术, 云数据管理技术, 大数据管理技术.

寿黎但(1976－), 男, 博士, 教授, 博士生导师, CCF 高级会员, 主要研究领域为非结构化数据管理, 移动社会媒体数据管理, 多媒体挖掘.



骆歆远(1988－), 男, 博士, 助理研究员, 主要研究领域为大数据管理, 大数据智能计算, 信息检索.

陈珂(1977－), 女, 博士, 副研究员, CCF 专业会员, 主要研究领域为非结构化数据管理, 数据挖掘, 隐私保护.



陈刚(1973－), 男, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为数据库, 大数据管理系统, 大数据智能计算.