软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2021,32(3):622−635 [doi: 10.13328/j.cnki.jos.006179]

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

数据库内 AI 模型优化*

钮泽平, 李国良

(清华大学 计算机科学与技术系,北京 100084)

通讯作者: 李国良, E-mail: liguoliang@tsinghua.edu.cn

摘 要: 在大量变化着的数据中,数据分析师常常只关心预测结果为特定值的少量数据.然而,利用机器学习模型进行推理的工作流程中,由于机器学习算法库默认数据以单表方式组织,用户必须先通过 SQL 语句查询出全部数据,即使随后在模型推理过程中会将大量数据丢弃.指出了在这个过程中,如果可以预先从模型中提取信息,就有望能在数据获取阶段快速排除不需要的数据,从而降低数据获取过程中的多表连接代价、进程间通信代价以及模型预测代价,进而加速整个工作流程.以决策树模型为例,首先提出一种预筛选+验证的执行方法对查询过程进行优化,之后给出了从决策树中提取用于预筛选谓词的离线算法,最后在真实数据集上进行测试.实验结果表明,所提出的方法能够对借助决策树模型推理结果对数据进行筛选的应用场景起到较好的加速效果.

关键词: SQL;数据库;决策树;DB4AI

中图法分类号: TP311

中文引用格式: 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

英文引用格式: Niu ZP, Li GL. In-database AI model optimization. Ruan Jian Xue Bao/Journal of Software, 2021,32(3):622−635 (in Chinese). http://www.jos.org.cn/1000-9825/6179.htm

In-database AI Model Optimization

NIU Ze-Ping, LI Guo-Liang

(Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China)

Abstract: In a large number of changing data, data analysts often only care about a small amount of data with specific prediction results. However, users must query all the data by SQL before inference step, even if a large amount of data will be dropped, because the machine learning algorithm libraries always assume that the data is organized in a single table. This study points out that in this process, if some hints can be gotten from model in advance, it is expected that unnecessary data can be quickly eliminated in the data acquisition phase, thus reducing the cost of multi-table join, inter-process communication, and model prediction. This work takes a specific kind of machine learning model, i.e., decision tree, as an example. Firstly, a pre-filtering and validation execution workflow is proposed. Then, an offline algorithm is used to extract pre-filtering predicates from the decision tree. Finally, the algorithm is tested on real world dataset. Experiments show that the method proposed in this study can accelerate the execution of SQL queries containing predicates on decision tree prediction result.

Key words: SQL; database; decision tree; DB4AI

近年来,数据量爆炸式增长.生产环境中,大部分数据都由数据库管理系统进行管理.机器学习算法目前广泛应用于各种数据分析与线上应用场景,然而在易用性方面,由于编写机器学习代码的复杂性且数据分析师更

* 基金项目: 国家自然科学基金(61925205, 61632016)
Foundation item: National Natural Science Foundation of China (61925205, 61632016)
本文由“支撑人工智能的数据管理与分析技术”专刊特约编辑陈雷教授、王宏志教授、童咏昕教授、高宏教授推荐.
收稿时间: 2020-07-20; 修改时间: 2020-09-03; 采用时间: 2020-11-06; jos 在线出版时间: 2021-01-21

熟悉 SQL 而存在较大提升空间;在执行效率方面,由于机器学习算法与数据库操作分离而无法借助数据库查询优化过程进行优化[1,2].数据库提供的 SQL 接口具有声明式的特点,用户通过 SQL 简练地指定查询任务,无需关注内部实现方法,数据库就可以通过一系列查询处理过程,将查询语句转化为一系列算子操作,利用索引、缓存等进行加速,选取合适的多表连接方案,确保准确而高效地获得数据.机器学习的训练与预测可以看作新的操作符,通过 SQL 支持这种新的操作符,一方面可以将数据获取与训练/预测过程通过 SQL 进行融合,使系统更易于使用;另一方面,提供了使用数据库优化机器学习算子的可能.已有的工作中,主要为通过数据库的用户定义函数(UDF)功能,对机器学习运算进行支持;或者通过结合硬件特点,利用数据库内联用户定义函数,将声明式查询与命令式数据处理转化为中间表示后统一优化等方式进行加速[2].

然而,在机器学习推理过程的优化中,前人的工作主要集中于通过查询语句中的谓词特点对模型进行剪枝,而对模型特征的利用仍不够彻底.考虑一个例子"选择周营业额预测值大于 20 万元的商店",可以通过 SQL 查询表示为:

SELECT *StoreName*
FROM *store*, *sales*, features AS *f*
WHERE (*store.id=sales.storeid* **AND** *sales.time=f.time*) **AND**
predict (*store.dept,f.CPI,...*)>200000;

在这类查询中,满足条件,需要返回给用户的数据可能只占全部数据中很小的比例.类似的例子还有:大型连锁超市希望知道各分店各销售部门中,营业额少于特定金额的有哪些;银行希望找出违约概率高于一定值的账户等.这些场景中,通过支持 UDF 的 SQL 可以表示为 WHERE 子句中包含机器学习预测函数的形式,即,使用机器学习模型预测结果对数据进行筛选可看作一种新的谓词.当数据量较大时,如果按照用户定义函数的方式进行执行,首先需要将多个包含特征列的表进行代价较为高昂的连接操作,之后必须在得到的全部数据上进行机器学习推理,才能得到占总数据量比例很小的查询结果.

然而,这种朴素的方式不一定是最优的.一方面,在未进行连接时,通过单表中某些重要特征的取值已经可以排除其在最终结果中的可能,但这种朴素的方式仍然会将连接操作进行到底,造成计算的浪费;另一方面,机器学习的推理过程与简单谓词相比复杂得多——较为复杂的决策树预测一条数据可能相当于十余条简单谓词的执行时间,故连接操作生成过多的无用结果,也会造成推理过程时间大幅增加.对于这类使用了机器学习推理结果对结果进行筛选的 SQL 查询,如果能够提前从模型中提取一些条件对数据预先进行筛选过滤,则可以同时减少数据获取中的连接代价与后续模型推理过程中的计算代价,从而提高执行效率.这样的规则需要是保守的,即不能因为应用规则导致应在最终结果中的数据被丢弃.直接进行连接操作得到的数据记为 S_{join},经过机器学习推理谓词过滤得到的结果记为 S_{result},使用规则筛选得到的数据记为 S_{filter},它们之间的关系可以表示为图 1.对于一个特定问题,S_{join} 与 S_{result} 越大,使用规则进行初筛的优化潜力就越大;而为了实际达到良好的加速效果,我们的目标便是让 S_{filter} 尽可能接近 S_{result}.

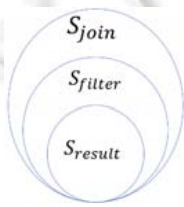

Fig.1 Relation of S_{join}, S_{filter} and S_{result}

图 1 S_{join},S_{filter},S_{result} 三者应满足的关系

用于初筛的规则一定与模型相关——只有模型才能够决定推理的结果,故我们可以尝试从以下两个角度切入.

(1) 构建更容易获得初筛规则的模型.机器学习模型中,各种特征重要性不同,如果我们能够在构建模型

过程中,在不显著损失预测准确率的情况下构建出能够让更多数据仅通过少量重要特征判断即可得到结果的模型,那么对这种更简单的模型放宽条件,有助于我们得到更接近 S_{result} 的 S_{filter};

(2) 对已有模型进行简化,提取结果数据的“大致特征”(使用谓词进行刻画)作为初筛规则.这里的谓词必须足够简单,因为过于复杂的谓词用于初筛会造成查询优化过程与执行过程中的性能损失.如果能得到满足上述特征的简单谓词,那么就可以借助数据库查询优化过程,下推用于初筛的简单谓词,降低数据获取过程中的多表连接开销;同时,初筛去掉大量结果后,需要执行推理的数据量也会显著降低.

本工作以机器学习中经典的决策树算法为例,通过 SQL 对决策树模型的训练方法以及包含决策树推理的谓词进行了支持,并设计算法进行性能优化.本文尝试利用数据库对一种特定的机器学习算法,决策树的推理过程进行优化.主要的贡献是:实现了通过 SQL 进行模型训练,加速模型推理的工作流程;提出通过预筛选+验证方法加速决策树推理 SQL 的速度,并设计了从决策树提取用于对查询结果预筛选的谓词算法;在真实数据集上测试加速效果,并分析实验结果.

本文第 1 节介绍使用数据库支持机器学习方向已有的工作.第 2 节以决策树模型为例,介绍了 AI 模型训练和推理 SQL 在数据库内的优化工作流程.第 3 节介绍数据库内决策树训练和推理的主要优化算法,针对前文提出的两个切入点进行尝试.第 4 节介绍在真实数据集上进行测试的效果并分析原因.第 5 节对其他 AI 模型上的优化进行探讨.最后,在第 6 节中进行总结.

1 相关工作

机器学习包括数据清洗与特征工程、模型训练、模型选择与评估、模型上线部署等一整套流程.消耗更少时间、得到更好的模型有利于提高数据分析师的效率,提升模型开发效率,降低应用系统处理延迟.近年来,数据库研究者对对机器学习整个流程的优化做出了诸多贡献(DB4AI),主要集中于扩展 SQL 语句支持机器学习算法、自动特征选取、模型训练加速、执行引擎优化方面.

- 声明式机器学习

为了能够降低机器学习算法的使用门槛,数据库社区出现了一些通过扩展 SQL 对机器学习算法进行支持的尝试.其思想为:通过声明式语言对复杂的机器学习算法执行过程进行封装,让用户能像使用 SQL 一样只需定义任务,系统便能自动实现细节,完成整个机器学习流程.MADlib[3]是开源的数据库内分析算法库,提供了基于 SQL 的机器学习算法,用户无需将数据导入导出其他工具即可完成机器学习任务.MADlib 的实现利用了 PostgreSQL 数据库的服务器端编程特性,将 C++,Python 编写的机器学习算法声明为用户定义函数.C++语言用于实现对性能要求较高的机器学习算法底层操作,如矩阵运算;Python 实现算法更高抽象层次上的逻辑.训练算法方面,MADlib 支持迭代式的训练过程;推理算法方面,MADlib 要求数据预先被连接为单表,推理结果亦通过表的方式返回给用户.MLlib[4]是 Spark 上原生的机器学习 API,它提供了快速、分布式的机器学习算法实现.MLlib 具有易于搭建机器学习全流程的 API,便于用户对从数据与处理、特征选取、训练模型、验证的全流程中各种方法进行替换.MLog[5]支持类似 SQL 的语法.MLog 将查询中机器学习运算与数据库运算分开,在 Tensorflow,Keras 等平台上通过 GPU 执行机器学习任务,在数据库中执行数据库运算.由于运算分开在不同平台执行,所以会有较大的通信开销.

- 自动特征选取

机器学习算法使用的数据需要包含足够的有用特征,尽量减少无关特征.给定一个用于训练机器学习任务的数据库,数据分析师需要耗费很多精力判断需要连接哪些表、是否需要通过额外的数据表增加特征.文献[6]首先提出,数据库中某些连接是可以避免的,他们提出数据库中一些表主键的值可以作为其中新特征的表示,避免与这些表的连接不会对模型的预测准确度产生影响.ARDA[7]借助用户给定的一个用于机器学习任务的数据基表和一个与该任务可能相关的大量数据表构成的数据仓库,自动发掘数据间的潜在连接关系,并寻找对机器学习任务帮助大的特征对数据集进行增强.

- 模型训练加速

(1) 通过分布式训练加速模型训练,其主要目标为提高并行度,降低通信中数据与模型的通信代价.LAPSE[8]将数据分布于各个计算节点,实现了在计算节点之间动态分配参数.ColumnSGD[9]提出将数据按列组织,将模型参数巧妙拆分,分配到各个计算节点,支持逻辑回归、SVM 和部分 DNN 模型;

(2) DB4ML[10]提出通过扩展 SQL 使其支持迭代事务时,传统的事务执行过于笨重,该工作针对不同机器学习算法使用不同隔离级别,调整迭代事务中 MVCC 记录方式,可以在满足训练要求条件下大幅提升训练效率.

- 执行引擎优化

机器学习算法中常会涉及到线性代数运算,对其执行进行优化可以提升训练和推理算法执行速度.LARA[11]提出一种新代数,在逻辑表示方式上融合了线性代数(LA)和关系代数(RA),从而获得了更多优化机会.SPORES[12]引入了对线性代数表达式的一种新的通用优化方法:构建一系列线性代数表达式和关系代数表达式之间相互转化的规则,借助规则先将线性代数表达式转换为关系代数表达式,对后者进行优化,再将结果转换回线性代数表达式.

当前,对模型推理进行加速的工作相对少.与本文举例介绍的通过数据库技术优化决策树模型推理较为接近的研究为文献[13],该工作介绍了 LinkedIn(领英)对职位搜索和推荐系统的优化,他们训练决策树学习文档特征到文档质量高低的映射,为了获得 top-k 文档,借助决策树叶子上包含样本正负例数目比例选取部分叶子到根路径上的条件得到新查询.本文在技术上虽然同样借助了决策树解释性强的特点从模型中提取信息,但本文并非解决 top-k 问题,从而引起从模型中提取的筛选谓词过多的问题,本文给出了谓词合并算法加以解决,并提出了通过初筛对含机器学习模型推理谓词的查询进行优化的总体框架.

2 数据库内 AI 模型工作流程

AI 算法通常包含训练与推理算法,需要通过设计 SQL 接口从语法上分别进行支持,SQL 接口的封装整合了 AI 算法与数据库查询操作,赋予 AI 算法声明式特点,同时也是借助数据库特性进行优化的前提.为了加速推理执行速度,在训练过程中,可以构建适合于数据库优化的模型,离线提取特征;在推理过程中,可以借助提取出的模型特征设计数据库算子和机器学习算子的联合优化.本文以决策树为例给出训练和推理过程中的优化技巧,整体工作流程如图 2 所示,本节将对其进行详细介绍.

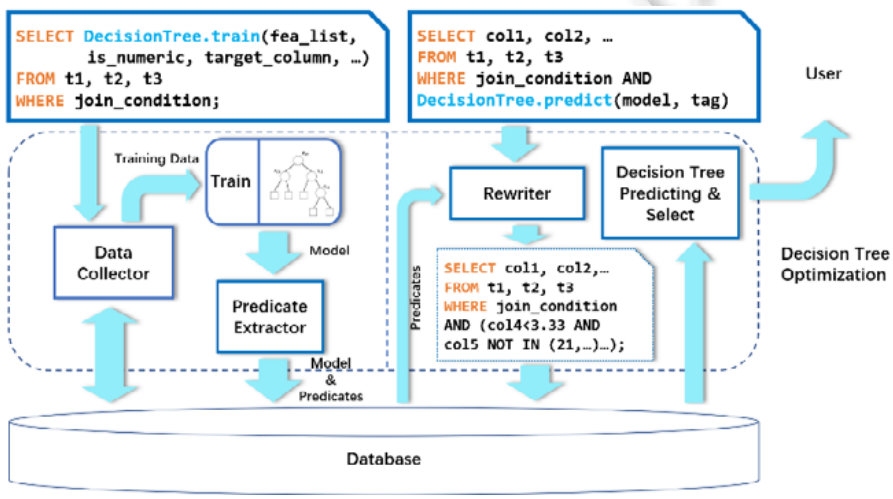

Fig.2 In-database decision tree optimization workflow

图 2 数据库内决策树优化工作流程

2.1 决策树训练SQL的执行过程

首先,用户输入定义模型的 SQL 语句,SELECT 子句中包含 *DecisionTree.train* 函数,其参数依次序分别指定参与训练过程的特征列表(*fea_list*)、各特征是定类特征或数值特征(*is_numeric*)、预测目标列名称(*target_column*),以及训练决策树需要的超参数(如使用 *gini* 系数或 cross entropy、决策树最大深度等).定义模型训练任务的 SQL 语句首先进入数据收集器(data collector),收集器会从 *Decision.train* 函数参数中提取出训练所需的特征列名,通过修改原 SQL 语法树的方式构造生成训练数据的 SQL 语句,进而通过数据库获取训练数据集.收集器获取到训练数据后,将训练数据与训练超参数传递给训练模块生成决策树.

本工作复现了 CART(classification and regression tree)算法[14]进行模型训练.在 CART 算法之外,亦尝试修改决策树训练算法.通过按照表的顺序建立决策树,借助后减枝[15]避免过拟合,在不显著降低模型准确度的情况下,获得更多只包含少量表特征的分支,从而降低推理过程的计算代价,并通过这种方式初步简化模型以利于提取谓词,相关部分见第 3.2 节.

训练后的决策树模型会被传递给谓词提取模块(predicate extractor).由于谓词提取过程较为耗时,且可以在进行推理前进行离线预处理,故我们会在训练得到决策树后马上进行谓词提取,并将提取出的谓词与原决策树模型一同存入数据库.决策树的叶子节点对应一个分类结果,根到叶子的路径上每个节点对应一个谓词,这些谓词组成的合取表达式可筛选出到达这个叶子节点的数据.然而,决策树模型的叶子数量可能十分庞大,如果用过多叶子节点对应的谓词共同组成的析取表达式直接生成 SQL 表达式,相当于每条数据需要对所有的决策树分支进行判断,在计算量上是不可接受的,所以谓词提取步骤是必要的.谓词提取器会对每个分类结果对应的谓词析取表达式进行合并,通过适当“放宽”条件,获得能够描述该类别主要特征的少量谓词.最终,提取后的谓词、决策树模型以及关于决策树使用的各特征类型(数值或者定类)将被存入到数据库中,同时将新创建的模型名称被返回给用户,以供包含决策树推理的 SQL 查询调用.

2.2 含决策树推理的SQL查询优化过程

用户希望借助训练好的决策树模型筛选预测值为特定值的数据时,可输入 WHERE 子句中包含决策树推理谓词的 SQL.

查询首先到达重写器(rewriter),重写器将查询转化为语法树,从决策树推理函数的参数中提取出模型名称与期望筛选的分类结果.借助模型名称,重写器从数据库中取出决策树模型、被提取出的预筛选谓词以及各列特征的类型信息.随后,重写器从预筛选谓词中取出筛选的分类目标对应的初筛条件,将其转化为谓词的语法树格式后,对决策树推理谓词进行整体替换.需要注意:当决策树推理谓词被 NOT 否定时,替换过程中有一些细节问题需要处理,将在第 3.4 中进行讨论.修改后的语法树被转换回用于初筛的 SQL,在数据库中进行执行得到初筛后的结果.初筛后的结果被送达验证模块.验证模块将初筛后的数据逐条通过决策树模型进行预测,筛除不符合用户输入语义的部分,将最终结果返回给用户.

3 数据库内决策树模型优化算法

本节会从引言部分最后给出的两个角度入手,引入并详细介绍上一节中决策树训练与推理优化过程中的关键算法.

3.1 通过谓词筛选执行决策树推理

决策树推理算法的输入是一棵决策树以及一组需要执行推理算法的数据,输出是这组数据中每条的分类结果.通常的推理算法需要对数据中的每个条目,从决策树根开始,根据是否满足节点所对应的谓词条件选择向左/右分支前进,直到走到叶子节点得到分类结果.然而,我们也有另一种策略可供选择,下面举例介绍.

例:医院需要用决策树模型获得预测手术后住院时间大于等于 10 天的患者.数据集包含属性:患者姓名,患者年龄,患者性别,手术类别,医生姓名,医生年龄,医生职称.训练后得到图 3 所示的决策树.

我们可以采用一种从结果逆推条件、在数据中进行选择的方式进行决策树推理.从决策树中我们可以看

出:满足条件的数据在逐条执行推理时,必定会到达值为 10,12,15 的叶子之一.到达值为 10 的叶子需满足(手术类别=类别 *A* AND 手术时长≥3h),记为 P_1;到达值为 12 的叶子需满足(手术类别=类别 *B* AND 患者年龄≥30 岁 AND 医生职称=主任),记为 P_2;到达值为 15 的叶子需满足(手术类别=类别 B AND 患者年龄≥30 岁 AND 医生职称=非主任),记为 P_3.所以我们可以通过条件(P_1 OR P_2 OR P_3)得到预测住院时间大于等于 10 天的患者.借助这种表示方式,P_2和P_3可以合并为(手术类别=类别*B* AND 患者年龄≥30 岁).对应于决策树上,也就是针对问题对决策树进行了简化.

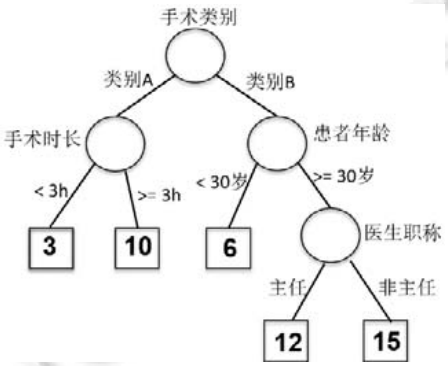

Fig.3 An example of decision tree for predicting length of stay

图 3 预测住院时间的决策树示例

3.2 决策树训练模型优化——按照表的顺序构建决策树

继续借助上文中的例子,需要进行推理的数据在医院数据库中以多表方式存储为 3 个关系表:患者(patient),手术(operation),医生(doctor),见表 1.

Table 1 Schema of tables in operation information database

表 1 患者手术信息数据库示例

表名	数据列
患者	PID(主键),姓名,年龄,性别
手术	OID(主键),患者 PID,主刀医生 DID,手术时长,手术类别
医生	DID(主键),姓名,年龄,职称

借助上文得到的筛选谓词,这个推理并筛选的过程可以表示为下面的 SQL 查询.

```
SELECT patient.id, patient.name
FROM patient p, operation op, doctor d
WHERE (p.pid=op.pid AND op.did=d.did)
  AND ((op.type='A' AND op.length≥3) OR
    (op.type='B' AND p.age≥30 AND d.title=1) OR
      (op.type='B' AND p.age≥30 AND d.title=0))
```

由于构建决策树时,我们的依据是每个特征将数据分割前后的信息增益,选择信息增益较大的特征,并没有考虑特征在表中的分布,所以提取出的叶子节点的谓词(如 P_1~P_3)中往往含有多个表中的特征.这样构造的 SQL 查询在进行查询优化时,由于被 OR 连接的谓词中分散着关于多个表的谓词,故难以进行谓词下推操作.

我们尝试在构建决策树时,初始只使用一张表中的特征,之后在构建好的树上,每个叶子节点作为新的根节点继续使用第 2 张表、第 3 张表、…,以此类推.在决策树生长中,使用预剪枝;在所有表都使用完毕后,使用代价-复杂性剪枝算法减小过拟合.希望借助这种建树方式得到更多只使用单表中的特征就可得到分类结果的分支,使生成的 SQL 更利于查询处理过程中的优化.

3.3 决策树模型推理优化——从决策树中获得预筛选谓词

第 3.1 节提出的谓词提取方式可能会导致过多叶子节点对应的谓词进行 OR 连接.这种情况下,由于 OR 连接的谓词有重叠的部分(如上面例子中,P_2 与 P_3 都满足 $op.type$='B'以及 $p.age\geqslant 30$),推理性能反而会下降.为了减少谓词的数量,同时便于查询优化中谓词下推,我们对直接通过叶子节点提取出的谓词进行合并,目标是得到包含较少谓词、易于进行谓词下推的 SQL 查询.

为了正式地描述上面的过程,我们首先引入叶子节点谓词的定义.

定义 1. 叶子节点对应的谓词,记为 P_{leaf},定义为决策树根到某个叶子节点经过分支条件 P_{leaf}^i 的合取,即:

$$P_{leaf}=\wedge_i P_{leaf}^i.$$

P_{leaf} 中可能存在关于相同特征列的 P_{leaf}^i 和 P_{leaf}^j,我们进行如下处理:当该特征是数值型时,进行表 2 中的转换;当该特征是定类的时,进行表 3 中的转换.经过整理后,各特征列最多在每个 P_{leaf} 中出现一次.在到达叶子节点路径上未进行选择的特征列不会出现在 P_{leaf} 中,我们使用值为 TRUE 的 P_{leaf} 项进行填充.至此,我们可以得到 $P_{leaf}=\wedge_{1\leqslant i\leqslant n}P_{leaf}^i$,其中,$n$ 为特征列的数目,P_{leaf}^i 为第 i 个特征列上的谓词或者常量 TRUE.

Table 2 Rules to transform predicates on numeric columns

表 2 数值谓词 $P_{leaf}^i\wedge P_{leaf}^j$ 整理规则

整理前	整理后
$f<c$	$-\inf<f<c$
$f\geqslant c$	$c\leqslant f<\inf$
$(c_1\leqslant f<c_2)\wedge(c_3\leqslant f<c_4)$	$\max(c_1,c_2)\leqslant f<\min(c_3,c_4)$

Table 3 Rules to transform predicates on categorical columns

表 3 定类谓词 $P_{leaf}^i\wedge P_{leaf}^j$ 整理规则

整理前	整理后
$f=c$	$f\in set\{c\}$
$f<>c$	$f\notin set\{c\}$
$(f\in set_1)\wedge(f\in set_2)$	$f\in set_1\cap set_2$
$f\in set_1\wedge f\notin set_2$	$f\in set_1-set_2$

3.3.1 谓词表达式的合并

所有在筛选范围内的结果对应的叶子节点记为 $leaves$,借助 P_{leaf},可以得到通过筛选进行推理的谓词为

$$\vee_{leaf\in leaves}P_{leaf}=\vee_{leaf\in leaves}(\wedge_{1\leqslant i\leqslant n}P_{leaf}^i).$$

如前文所述,#{$leaves$}可能相当多,我们不应将如此多 P_{leaf} 组成的析取式放到 SQL 中进行执行.注意到:不同 P_{leaf}^i 对于相同的 i,都是关于第 i 个特征列的谓词,通过对不同叶子关于相同特征列谓词进行合并,我们可以将谓词降低到特征列数级别.

命题 1. $\vee_{leaf\in leaves}(\wedge_{1\leqslant i\leqslant n}P_{leaf}^i)$ 筛选数据得到的结果集合记为 R, $\wedge_{1\leqslant i\leqslant n}(\vee_{leaf\in leaves}P_{leaf}^i)$ 筛选数据得到的结果集合记为 S,则 $R\subset S$.

命题 1 是我们进行谓词合并的依据.通过更改析取和合取操作的顺序,相同特征的谓词可以通过析取运算进行合并,命题保证了合并后谓词筛选得到的数据不会缺少原先谓词筛选出的数据,所以这种合并是保守且可行的.这个命题可以通过数学归纳法证明.

证明:对特征列数目 n 归纳,记叶子节点数#{$leaves$}为 m.

(1) n=1 或 m=1,转换前后形式相同,为平凡情况 n=2,m>1 时有:

$$\vee_{1\leqslant j\leqslant m}(P_j^1\wedge P_j^2)=(\vee_{1\leqslant j\leqslant m}P_j^1)\wedge(\vee_{1\leqslant j\leqslant m}P_j^2)\wedge(\vee_{1\leqslant j\leqslant m,i\in\{1,2\}}P_j^i).$$

故 $n\leqslant 2$ 时命题成立;

(2) 假设 $n=n_0-1$ 时命题 1 成立;

(3) 当 $n=n_0$ 时,$\vee_{1\leqslant j\leqslant m}(\wedge_{1\leqslant i\leqslant n_0}P_j^i)=\vee_{1\leqslant j\leqslant m}(P_j^{n_0}\wedge(\wedge_{1\leqslant i\leqslant n_0-1}P_j^i))$ 通过 $n=2$ 的情况以及情形(2)中的归纳假设,可知对 $n=n_0$ 时也成立. □

据此,对于同特征列上的谓词,我们通过 OR 运算进行合并.规则如下.

数值特征. 对于数值特征,P_{leaf}^i 均为区间形式,计算这些谓词的析取即为计算这些区间的并集.可以通过贪心的算法进行:按照区间左端点进行排序后进行一次遍历,遍历中临近的区间贪心地扩展.

定类特征. 对于定类特征,P_{leaf}^i 的形式为 $feature\in S$ 或者 $feature\notin S$,可根据表 4 进行化简.

Table 4 Rules to merge predicates on categorical columns

表 4 定类谓词 $P_{leaf_a}^i\wedge P_{leaf_b}^i$ 合并规则

整理前	整理后
$(f\in set_1)\vee$TRUE	TRUE
$(f\notin set_1)\vee$TRUE	TRUE
$(f\in set_1)\vee(f\in set_2)$	$f\in set_1\cup set_2$
$(f\notin set_1)\vee(f\notin set_2)$	$f\notin set_1\cap set_2$
$(f\in set_1)\vee(f\notin set_2)$	$f\notin set_2-set_1$

3.3.2 避免谓词被全部抵消

然而,上述合并方式仍存在一定问题:如果每个特征 i 都存在 $P_{leaf}^i=$TRUE 的叶子,那么最终全部谓词都会被消去.为了能够尽可能避免这种情况,并且仍能够进行谓词下推,我们从以特征为单位进行合并转为以表为单位进行合并.下面用一个简单的例子说明.

如表 5 所示,有两个关系表 T_1 和 T_2 以及 4 个特征 f_1~f_4,每一行是一个叶子上对应的谓词.

Table 5 An example in which predicates are all eliminated after merging

表 5 谓词全部被消除情况示例

	$T_1.f_1$	$T_1.f_2$	$T_2.f_3$	$T_2.f_4$
$leaf_1$	P_1^1	P_1^2	TRUE	P_1^4
$leaf_2$	P_2^1	TRUE	P_2^3	P_2^4
$leaf_3$	P_3^1	P_3^2	P_3^3	TRUE
$leaf_4$	TRUE	P_4^2	TRUE	P_4^4
$leaf_5$	TRUE	P_5^2	P_5^3	P_5^4
$leaf_6$	P_6^1	P_6^2	P_6^3	TRUE

表中每一列均包含 TRUE,进行第 3.3.1 节中的合并后只能留下 TRUE,所有能进行数据筛选的谓词全部被消除了.然而我们发现:如果以关系表为单位考虑,则不存在使 T_1 或者 T_2 中关于所有特征的谓词同时为 TRUE 的叶子.所以我们可以先选择一个特征,将叶子节点按照在这个特征上的谓词是否为常量 TRUE 分为两部分,对为 TRUE 的部分使用其他特征的谓词进行描述.即,上表中的谓词可合并为

$$((P_1^1\vee P_2^1\vee P_3^1\vee P_6^1)\vee(P_4^2\vee P_5^2))\wedge((P_2^3\vee P_3^3\vee P_5^3\vee P_6^3)\vee(P_1^4\vee P_4^4)).$$

这样合并后,不同关系表的特征不会通过析取进行连接,依然可以直接将各个表的谓词表达式下推到底.

3.4 决策树推理谓词取反的情况

上文的谓词合并过程默认了决策树推理谓词没有被否定运算 NOT 修饰.比较隐蔽的一点是:如果决策树推理谓词被 NOT 修饰,而我们直接将原谓词替换为模型提取得到的预筛选表达式,那么我们会丢失很多本应在结果中的数据.这是因为:若全部数据为 U,使用决策树推理谓词筛选得到的数据集合为 S_{tree},使用提取出的初筛谓词筛选得到的数据集合为 S_{filter},那么用决策树推理谓词的否定进行筛选得到的数据集合为 $U\backslash S_{tree}$,使用初筛谓词的否定筛选出的数据集合为 $U\backslash S_{filter}$.由于 $S_{filter}\supseteq S_{tree}$,所以有 $S_{result}=U\backslash S_{tree}\supseteq U\backslash S_{filter}$,即:替换后,筛选得到的数据

丢失了部分本应包含在结果中的数据.这种情况的解决亦不困难:设决策树全部叶子节点集合为 S_{leaves},根据决策树推理谓词得到的叶子集合为 S_{target},那么当发现推理谓词被 NOT 修饰时,使用 $S_{leaves}-S_{target}$ 的叶子节点集合对应的谓词进行合并,便可保证初筛过程不会遗漏最终结果.

4 实验与分析

4.1 实验设置

4.1.1 数据集介绍

本工作使用 Kaggle 上公开的 Walmart 销量预测数据集(https://www.kaggle.com/c/walmart-recruiting-store-sales-forecasting)进行实验与性能测试.该数据集由 3 个数据表构成,分别是 Sales(预测目标周销量),Features(各商店所在地区,各周反映经济状况、天气等的参数),Store(超市大小,类型).3 个数据表的连接关系如图 4 所示,图中数据表后的括号中为数据条数.

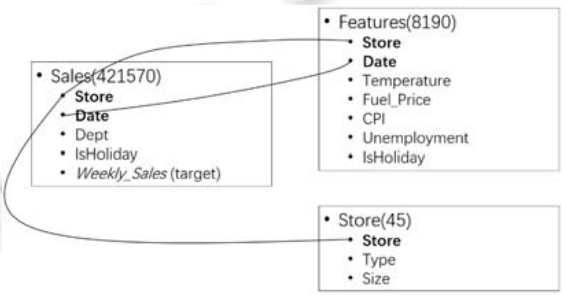

Fig.4 Join graph of Walmart sales prediction dataset

图 4 Walmart 销量预测数据集连接关系

原数据中,预测目标 *Weekly_Sales* 是浮点数,为了更好地适应决策树算法,以及研究本文中算法关于结果数据量多少的运行效率,我们对 *Weekly_Sales* 进行等深分箱,并按照销量从小到大,离散化为等级 1,2,…,k.原数据中部分属性值缺失,我们提前使用平均值进行填充.

4.1.2 实验环境与实现细节

本工作使用 PostgreSQL 12.2 数据库,将 *Sales*,*Features*,*Store* 这 3 个数据表导入.测试用 SQL 查询通过决策树推理谓词筛选达到某个销量等级的商店与日期,形式如下.

SELECT *s.store, s.dept, s.date*
FROM *sales s, features f, store st*
WHERE *DecisionTree.predict*('*cart_weekly_sale*',5)
  AND (*s.store=f.store* AND *s.date=f.date* AND *s.store=st.store*);

实验代码使用 python 编写,借助 moz_sql_parser(https://github.com/mozilla/moz-sql-parser)进行 SQL 查询语法解析.实验中,决策树训练使用 *Gini* 指数作为选取属性的指标,按表顺序构建决策树时人为指定了顺序(*Sales*, *Store*,*Features*).

4.1.3 评测指标

关于按表顺序建树得到的模型与 CART 决策树模型的对比,本工作使用模型大小(叶子节点总数)、叶子节点谓词少于 3 个表的数目、在测试集上预测准确率进行评价.

对决策树推理谓词优化效果最直观的评价指标是推理查询执行时间.本工作对比以下几个执行时间:从输入 SQL 到连接数据或初筛后数据返回的时间 t_{join},主要受单表扫描与多表连接用时影响;使用决策树模型进行

决策树推理过程所用时间 $t_{predict}$;总用时 t_{total}.

PostgreSQL 支持通过添加 EXPLAIN(ANALYZE ON)获得实际执行计划,本工作通过对比加入初筛谓词前后执行计划变化分析性能提升原因.

4.2 优化效果

4.2.1 执行时间测试

图 5 为是否使用决策树模型中提取出的初筛谓词对数据获取 SQL 在数据库中执行效率的影响.横坐标为分类数目,由于使用了等深分箱,测试用 SQL 筛选其中一个类别,故分类数 k 越大,最终结果中数据越少,从图中可看出加速数据筛选过程越好,其中,k=20 时的加速比可以达到 43%.表明谓词被下推到单表,减小了连接运算计算量.

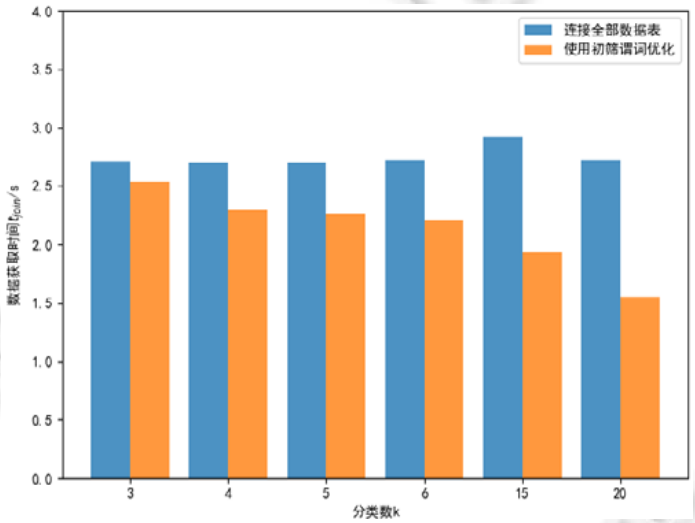

Fig.5 Data fetch time comparision between with/without using extracted predicates

图 5 初筛谓词对数据获取过程的加速效果

表 6 中列出了结果数据百分比(实验中通过分类数控制)、初筛过滤的数据占比、初筛+验证对决策树推理谓词加速比之间的关系.可以看出:初筛过滤移除的数据越多,整体加速效果越好,二者数值上也十分接近.

Table 6 Relationship between speedup ration and proportion of result data

表 6 加速比与结果数据百分比关系

结果数据百分比(%)	初筛过滤数据百分比(%)	加速比(%)
33.3	9.8	4.8
25.0	20.0	17.9
20.0	21.2	13.7
16.7	23.8	18.5
6.7	43.6	37.1

4.2.2 从执行计划分析性能提升

分类数目 k=15 时,图 6 与图 7 分别为在获取数据 SQL 中是否加入初筛谓词得到的两个执行计划,各个执行节点中,*actual_time* 表示实际实行消耗的时间,格式为“准备时间..执行时间”.从执行计划中可以看出,使用了初筛谓词时,对 *sales* 表进行扫描的过程中筛掉了 183 942 条数据,用时 226.513ms;未使用初筛谓词时,对 *sales* 表进行线性扫描耗时 63.202ms,故使用初筛谓词首先会增加扫描数据表的代价增加.*sales* 与 *features* 表哈希连接前,对 *features* 表进行了线性扫描后哈希,这个步骤的性能与是否使用初筛谓词无关,二者用时分别为 6.069ms

与 5.837ms.对于 *sales* 与 *features* 表进行哈希连接的代价,可以通过总时间减去 *sales* 表的线性扫描以及 *features* 表的线性扫描并哈希的时间,使用初筛谓词用时 125.976ms,未使用初筛谓词为 233.789ms.同理可计算得到最后与 *store* 表进行连接时,使用初筛谓词用时 83.686ms,未使用初筛谓词用时 114.258ms.验证了初筛谓词会加速连接操作的想法.

```
Hash Join (actual time=6.213..442.304 rows=237628 loops=1)
   Hash Cond: (s.store = st.store)
   ->  Hash Join (actual time=6.139..358.558 rows=237628 loops=1)
         Hash Cond: ((s.store = f.store) AND (s.date_time = f.date_time))
         ->  Seq Scan on sales s (actual time=0.032..226.513 rows=237628 loops=1)
               Filter: (dept <> ALL ('{12,19,20,…,98}'::integer[]))
               Rows Removed by Filter: 183942
         ->  Hash (actual time=6.068..6.069 rows=8190 loops=1)
               ->  Seq Scan on features f (actual time=0.012..2.642 rows=8190 loops=1)
   ->  Hash (actual time=0.060..0.060 rows=45 loops=1)
         ->  Seq Scan on stores st (actual time=0.017..0.028 rows=45 loops=1)
```

Fig.6 Execution plan of SQL with extracted predicates

图 6 使用初筛谓词的执行计划

```
Hash Join (actual time=6.056..447.217 rows=421570 loops=1)
   Hash Cond: (s.store = st.store)
   ->  Hash Join (actual time=5.920..302.828 rows=421570 loops=1)
         Hash Cond: ((s.store = f.store) AND (s.date_time = f.date_time))
         ->  Seq Scan on sales s (actual time=0.041..63.202 rows=421570 loops=1)
         ->  Hash (actual time=5.837..5.837 rows=8190 loops=1)
               ->  Seq Scan on features f (actual time=0.013..2.522 rows=8190 loops=1)
   ->  Hash (actual time=0.104..0.104 rows=45 loops=1)
         ->  Seq Scan on stores st (actual time=0.050..0.061 rows=45 loops=1)
```

Fig.7 Execution plan of SQL without extracted predicates

图 7 未使用初筛谓词的执行计划

从执行计划可以看出:使用了初筛谓词后,会增加线性扫描表时的代价,降低连接操作的代价.在上面的例子中,使用初筛谓词时增加的代价与减小的代价近乎抵消,最终只比未使用初筛谓词快大约 5ms,在系统波动范围内可以视作无差别.

然而实际测试中,k=15 时,使用初筛谓词获取数据过程的加速比达到了 33.9%.这是因为实验中优化过程作为 PostgreSQL 的扩展部分,为独立的进程,获取数据过程中会产生进程间数据拷贝的代价,而这个代价与传输数据量成正比.故使用初筛谓词的算法由于过滤掉了 44%的数据,使它的数据通信代价大幅减小,最终在数据获取阶段优于连接全部数据.

对使用/不使用初筛谓词的执行方法,在获得数据后,均需要进行决策树推理.由于二者的决策树相同,这个过程的执行时间依赖于需要进行推理的数据条数,故使用谓词初筛后进行推理计算的计算量小于不进行初筛的方法.表 7 支持这个观点.

Table 7 Influence of prefilter predicates on inference execution time

表 7 初筛谓词对推理执行时间的影响

初筛过滤数据百分比(%)	未初筛推理用时(s)	初筛后推理用时(s)	加速比(%)
9.8	7.4	7.1	4.3
20.0	7.3	5.9	19.1
21.2	7.5	6.5	12.8
23.8	7.8	6.3	18.4
43.6	8.8	5.5	38.2
50.4	9.0	5.0	44.4

4.3 构建决策树方法的影响

表 8 中,按表顺序构建决策树时,其与 CART 决策树的预测准确度是近似相同的,未造成推理误差显著提高.

Table 8 Accurcy comparision between table-order constructed tree and CART

表 8 按表顺序建树与 CART 决策树准确率比较

分类数目 k	CART Acc(%)	按表顺序建树 Acc(%)
2	93.31	93.04
3	87.53	86.13
4	91.97	84.41
5	78.38	81.26

按表顺序建树的目的,是让训练得到的决策树模型更多分支上只含较少表中的特征,然而实际测试发现效果较为有限.表 9 中,按表顺序建树后,包含更少表的叶子谓词增加量不大,绝大多数叶子对应的谓词仍然与全部 3 个数据表中的特征相关.

Table 9 Difference of extracted predicates between CART and table-order constructed tree

表 9 决策树叶子按照对应谓词相关的数据表统计数量

叶子谓词相关的表	CART (k=3)	Tab (k=3)	CART (k=4)	Tab (k=4)	CART (k=5)	Tab (k=5)
sales	0	9	0	10	0	8
sales,store	48	257	43	388	35	380
sales,features	0	0	0	19	0	3
sales,features,store	23679	20197	32850	33241	40555	41290
总计	23727	20463	32893	34658	40590	41681

模型的结构变化不大,暗示了这种构建决策树的方式对查询推理的加速效果十分有限.并且从叶子总数可以看出,按表顺序建树通常会造成模型的叶子数目增加,即模型复杂度增加,这对验证过程的执行效率会产生负面影响.从表 10 可以看出:按表顺序构建出的决策树执行推理查询时,性能没有明显优于 CART 算法构建出的决策树.

Table 10 Influence of decision tree construction method on query execution time

表 10 按表顺序建立决策树对查询执行效率的影响

	CART (k=3)	Tab (k=3)	CART (k=4)	Tab (k=4)	CART (k=5)	Tab (k=5)
数据获取时间(s)	2.8	2.8	2.3	2.4	2.3	2.3
验证时间(s)	5.9	5.7	5.2	5.3	5.7	5.3
总时间(s)	8.7	8.5	7.5	7.7	8.0	7.6

5 其他 AI 模型推理优化

本文提出初筛方法中使用的初筛谓词需要从模型中提取,决策树模型优化中的关键步骤——提取根节点到叶子节点路径上的谓词与其强解释性密切相关.对于其他 AI 模型,机器学习解释性领域有一些在局部通过强解释性模型对弱解释性模型进行拟合的方法[16,17]可以带来启发.机器学习解释性领域对于模型整体解释性主要考虑方便人类进行理解,而选取若干有代表性的单个预测实例的解释作为对模型的整体解释[16].而为了从模型中提取各类别特征需要更强的整体解释方法,由于从模型提取谓词是离线进行的,可以考虑对特征空间中进行适度采样以构建局部和整体解释的联系,这方面仍有待深入探索.

当前,机器学习领域也有结合规则增强深度模型解释性的尝试.文献[18]中提出了一种基于规则的层次化模型,并给出了对应的神经网络结构和对应的新训练方法,最终在多个数据集上准确度均能优于逻辑回归、SVM、MLP、GBDT 等模型.我们认为:在这类新的具有较强解释性和较高精度的模型上,本文提出的框架有推广潜力;且随着机器学习领域对解释性的重视增加,有望在未来见到更多这类模型.

6 总结与未来工作

本工作提出了通过预筛选+验证对 AI 模型推理进行优化的框架,举例进行了决策树模型优化,并对其他解释性较弱的模型优化思路进行了探讨.本工作通过扩展 SQL 支持了决策树训练与推理,并借助框架对决策树推理函数出现在谓词中的查询进行了优化.为了实现预筛选,本工作给出了从决策树中提取谓词并进行适当合并的算法.此外,本工作也尝试设计了按照表顺序构建决策树的训练算法,其预测误差与标准的决策树算法没有显著差别.在 Walmart 销量预测任务上,上述算法达到了提高包含决策树推理谓词的 SQL 执行效率的目标.

本工作也引起了我们一些新的思考,未来工作将从以下几个方面展开:首先,对于本文提出的谓词提取与合并算法,谓词数量和谓词筛选效果的权衡中,很可能存在更好的方案有待发现;第二,对决策树外的机器学习模型,通过构建局部解释性和模型全局特点的联系来设计模型无关的谓词提取算法;最后,针对新的解释性较强的模型,设计新的谓词提取算法.

References:

[1] 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

[2] Li GL, Zhou XH, Sun J, *et al*. A survey of machine-learning-based database techniques. Chinese Journal of Computers, 2019 (in Chinese with English abstract).

[3] Hellerstein JM, Re C, Schoppmann F, *et al*. The madlib analytics library or MAD skills, the SQL. Proc. of the VLDB Endowment, 2012, 5(12):1700−1711. http://vldb.org/pvldb/vol5/p1700_joehellerstein_vldb2012.pdf [doi: 10.14778/2367502.2367510]

[4] Meng X, Bradley JK, Yavuz B, *et al*. Mllib: Machine learning in apache spark. Journal of Machine Learning Research, 2016,17: 34:1−34:7. http://jmlr.org/papers/v17/15-237.html

[5] Li X, Cui B, CHEN Y, *et al*. Mlog: Towards declarative in-database machine learning. Proc. of the VLDB Endowment, 2017,10(12): 1933−1936. http://www.vldb.org/pvldb/vol10/p1933-zhang.pdf. [doi: 10.14778/3137765.3137812]

[6] Kumar A, Naughton J, Patel JM, *et al*. To join or not to join? Thinking twice about joins before feature selection. In: Proc. of the 2016 Int'l Conf. on Management of Data. 19−34. ACM, 2016.

[7] Chepurko N, Marcus R, Zgraggen E, *et al*. ARDA: Automatic relational data augmentation for machine learning. Proc. of the VLDB Endowment, 2020,13(9):1373−1387. http://www.vldb.org/pvldb/vol13/p1373-chepurko.pdf

[8] Renz-Wieland A, Gemulla R, Zeuch S, *et al*. Dynamic parameter allocation in parameter servers. Proc. of the VLDB Endowment, 2020,13(11):1877−1890. http://www.vldb.org/pvldb/vol13/p1877-renz-wieland.pdf

[9] Zhang Z, Wu W, Jiang J, *et al*. ColumnSGD: A column-oriented framework for distributed stochastic gradient descent. In: Proc. of the ICDE. 2020. 1513−1524. [doi: 10.1109/ICDE48307.2020.00134]

[10] Jasny M, Ziegler T, Kraska T, *et al*. DB4ML—An in-memory database kernel with machine learning support. In: Proc. of the 2020 Int'l Conf. on Management of Data (SIGMOD Conf. 2020). 2020. 159−173. [doi: 10.1145/3318464.3380575]

[11] Hutchison D, Howe B, Suciu D. LaraDB: A minimalist kernel for linear and relational algebra computation. In: Proc. of the 4th ACM SIGMOD Workshop on Algorithms and Systems for MapReduce and Beyond (BeyondMR@SIGMOD). 2017. [doi: 10.1145/3070607.3070608]

[12] Wang YR, Hutchison S, Leang J, *et al*. SPORES: Sum-product optimization via relational equality saturation for large scale linear algebra. Proc. of the VLDB Endowment, 2020,13(11):1919−1932. http://www.vldb.org/pvldb/vol13/p1919-wang.pdf

[13] Grover A, Arya D, Venkataraman G. Latency reduction via dcision tree based query construction. In: Proc. of the 2017 ACM on Conf. on Information and Knowledge Management (CIKM 2017). 2017. 1399−1407. [doi: 10.1145/3132847.3132865]

[14] Breiman L, Friedman JH, Olshen RA, *et al*. Classification and Regression Trees. 1984.

[15] Bradford JP, Kunz C, Kohavi R, *et al*. Pruning decision trees with misclassification costs. In: Proc. of the 10th European Conf. on Machine Learning (ECML'98). LNCS 1398, Chemnitz: Springer-Verlag, 1998. 131−136. https://doi.org/10.1007/BFb0026682

[16] Ribeiro MT, Singh S, Guestrin C. "Why should I trust you?" Explaining the predictions of any classifier. In: Proc. of the 22nd ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. 2016. 1135−1144.

[17] Lundberg SM, Lee SI. A unified approach to interpreting model predictions. In: Proc. of the Advances in Neural Information Processing Systems. 2017. 4765–4774. http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf

[18] Wang Z, Zhang W, Liu N, *et al*. Transparent classification with multilayer logical perceptrons and random Binarization. In: Proc. of the AAAI 2020. 2020. 6331–6339.

附中文参考文献:

[1] 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

[2] 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,2019.

钮泽平(1997－),男,博士生,CCF 学生会员,主要研究领域为数据库与机器学习的交叉技术.

李国良(1981－),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,大数据分析和挖掘,群体计算.