Baran:通过统一上下文表示和迁移学习进行有效的错误校正

09/24/2024-09/29/2024

摘要

传统的错误校正解决方案利用辅助规则或主数据来寻找正确的值。在实际应用中,这两种方法经常出错。因此,最好从有限数量的修复实例中学习校正方法。为了有效地归纳修复实例,有必要捕捉每个错误值的整个上下文。上下文包括值本身、同一元组中同时出现的值以及定义属性类型的所有值。通常,基于这些上下文信息的纠错器会经历一个单独的操作过程,这个过程并不总是容易与其他类型的纠错器集成。在论文中,我们提出了一种新的纠错系统——Baran,它为集成多个纠错器模型提供了一个统一的抽象,这些模型可以以相同的方式进行预训练和更新。由于我们方法的整体性,我们生成的校正候选比最先进的要多,而且由于我们使用了基于上下文的感知数据表示,我们实现了高精度。我们展示了通过基于维基百科修订版预训练我们的模型,我们的系统可以进一步提高其整体精度和召回率。在我们的实验中,Baran 在有效性和人工参与方面都明显优于最先进的纠错系统,只需要 20 个标记的元组。

1. 引言

数据清理是数据科学家最重要但最耗时的任务之一[14]。数据清理任务包括两个主要步骤:(1)错误检测和(2)错误校正(即数据修复)。错误检测的目的是识别错误的数据值[3]。错误校正的目的是将这些错误值修正为正确的值[37]。

传统的数据清理系统遵循预先配置范式,用户必须事先提供正确且完整的规则和参数集,例如功能依赖性和所需的数据模式[3, 37, 11, 13, 48]。对于大多数非专业用户而言,这是一个主要障碍,因为他们需要事先了解数据集和数据清理系统,以便能够正确配置系统[3, 27, 48]。最近,我们推广了免配置模式,用户只需标注几个数据值即可[27]。事实上,用户只需提供几个数据错误及其更正的示例,系统就会学会进行错误检测/更正操作。这种模式更适合以下三种情况: (1)数据集是全新的,没有数据约束;(2)用户是领域内的专家,不擅长为复杂的数据清理系统生成正确的规则/参数;(3)他们更喜欢额外标注一些数据值,以提高传统数据清理系统的性能。

由于我们的无配置错误检测系统 Raha [27]的前景广阔,我们在此研究将同样的直觉应用于后续的错误校正任务。无配置错误校正比其错误检测的对应任务更具挑战性。错误检测是一个二元分类任务,数据集中的每个值都有两个可能的类别,即干净或脏。然而,理论上任何可能的字符串都可以作为错误值的修正,因此纠错的类别空间是无限的。因此,我们往往有很大概率从该无限搜索空间中选择了错误的修正,进而影响纠错的精度。为了解决这个问题,以前的方法通过只考虑数据集本身[48, 37]或可信的外部来源[12]中的值来限制可能的修正空间。这些删减策略限制了错误校正的召回率,因为实际校正可能不存在于提供的资源中。

我们的方法。我们提出了一种新的无配置错误校正系统——Baran,该系统能够同时实现高精度和召回率。为此,Baran 结合了所谓的错误校正器模型,这些模型能够捕获错误数据周围的上下文信息。对于已经检测到数据错误的脏数据集,Baran 通过新颖的两步式错误校正任务来修复数据错误。首先,每个错误校正器模型为每个检测到的数据错误生成一组初始的潜在校正。这一步尤其能够提高纠错的可实现召回率。然后,Baran 将这些模型的输出以半监督的方式整合为每个数据错误的最终校正。事实上,Baran 迭代性地要求用户标记数据集中存在数据错误的元组,并利用用户提供的校正来逐步更新校正器模型。Baran 为每列训练一个分类器,从而为每个数据错误在所有校正候选中识别出最准确的校正。这一步骤尤其能够保持纠错的高精确度。

为了设计一套完整的纠错模型,我们需要利用所有数据错误上下文。原则上,为了修正给定数据集中的数据错误,我们可以利用三种数据错误上下文:

1. 数据错误的价值。有些数据错误仅通过考虑错误值本身即可修复。在这种情况下,我们只需将错误的数据值转换为正确的值即可。例如,错误的日期值“16/11/1990”可以通过将其转换为正确的格式(即“16.11.1990”)来修复。

2. 数据错误的位置。纠正一些数据错误需要了解其位置,即同一数据行中其他正确数据值的信息。例如,我们不能修复首都列中错误的“巴黎”值,因为数据值本身并不是一个错误值。但是,如果我们检查同一元组中相邻的干净数据值,并观察国家列中的“德国”,那么我们可以将“巴黎”改为“柏林”,使其与相邻值保持一致。

3. 数据错误域。修复一些数据错误需要域信息,即同一列中其他干净数据值的信息。例如,要修复一个异常的温度值,我们可以使用同一列温度中的其他干净值来推断正确的

值。

如上例所示,每个数据错误环境都可以通过完全不同的方式来生成修正候选值。这些修正程序虽然可以独立得出相同的修正候选值,但整合起来并不容易。

我们两步任务制定的另一个好处是,无需用户手动操作,即可对纠错器模型进行预训练,从而扩展可能的校正空间。事实上,Baran 还可以利用迁移学习,即从一项任务中获取知识,然后将其应用于不同但相关的任务[31]。当训练数据在一个领域中稀缺且昂贵,而在类似领域中广泛可用时,迁移学习就非常合适。在这种情况下,可以在相关领域对学习模型进行预训练,然后在当前数据集上进行微调[15]。在我们的示例驱动错误校正任务中,情况也是如此。除了当前数据集的干净值之外,我们需要更多的校正候选值来提高可实现的召回率。这些校正候选值必须由用户提供或从外部来源学习。虽然用户标注当前数据集的成本很高,但基于值的校正在外部来源中很常见,例如维基百科页面修订历史。因此,我们可以从外部来源提取价值更新,作为额外的校正示例来预训练错误校正器模型。预训练旨在学习常见错别字和错误的校正,例如将“Holland”误写为“Netherlands”。然后,可以在用户标签的帮助下,对预训练的模型进行微调。因此,Baran 还可以学习特定数据集的偏好,并在所有校正候选中选出最佳校正。我们利用维基百科页面修订历史作为丰富的数据源,其中包含人类提交的修订内容,数量达数百万兆字节。实际上,我们可以用同样的方法利用任何其他数据更新源。

挑战。为了设计这样的纠错系统,我们解决了以下问题:

- 如何设计可统一和增量更新的纠错器模型,为不同的数据错误提供各种可能的纠正方案?

- 我们如何为纠错制定半监督分类任务,从而在众多候选方案中有效且高效地确定实际纠错方案?

- 我们如何从公开可用的修订数据历史中提取基于价值的纠错,并使用它们预训练纠错器模型?

贡献。为了应对这些挑战,我们做出了以下贡献:

- 我们提出了一种新的无配置错误校正系统,名为 Baran,无需任何用户提供的规则或参数即可修正数据错误(第 3 节)。

- 我们提出了简单、通用且可逐步更新的错误校正模型(第 4 节)。这些模型利用数据错误的价值、邻近性和域上下文,为任何数据错误提出修正候选。

- 我们设计了一种新颖的二元分类任务,该任务得益于密集特征表示和元组采样方法,该方法选择最有用的元组来概括用户更正(第 4 节)。该任务有效地将错误校正器模型的输出组合为每个数据错误的最终校正。

- 我们提出了一种分割和调整维基百科页面修订历史的方法,以收集额外的训练数据,用于预训练基于价值的模型(第 5 节)。

- 我们进行了大量实验,从有效性、效率和人工参与度等方面评估我们的系统(第 6 节)。实验结果表明,在 7 个知名数据集上,Baran 系统明显优于 4 个最新的纠错系统。

2. 基础

我们首先正式定义了问题陈述,然后回顾了最先进的解决方案及其局限性。

2.1 问题陈述

纠错问题是指用相应的正确值替换检测到的数据错误。设 $d = \{t_1, t_2, \ldots, t_{|d|}\}$,大小为$|d|$的关系数据集,其中每个 t_i 表示一个元组。设 $A = \{a_1, a_2, \ldots, a_{|A|}\}$为数据集 d 的架构,具有$|A|$个属性。我们将 $d[i,j]$表示为元组 t_i 和属性 a_j 中的数据值。设 d^*为同一数据集的地面真实值,仅包含干净值。

数据错误是指数据集中的数据值与不可用的真实值存在偏差。我们主要考虑语法和语义数据错误类别[27]及其后续数据错误类型,例如缺失值、错别字、格式问题以及违反属性依赖关系[36]。令 $E = \{d[i,j] \mid d[i, j] \neq d^*[i,j]\}$为检测到的数据错误集,把它作为上游错误检测步骤的输出。检测到的数据错误的质量通常会影响下游的错误校正步骤。与之前的工作类似,我们基于错误检测步骤正确且完整的假设来讨论我们的方法[37, 48]。在我们的实验(第 6.6 节)中,我们还评估了 Raha[27]之上的系统性能,该系统提供的错误检测结果并不完美。

给定一个脏数据集 d、一组检测到的数据错误 E 以及一组用于标注元组的标签预算 θLabels, 我们的目标是尽可能多地修复检测到的数据错误。

2.2 当前技术水平

当前最先进的错误校正系统遵循预先配置的范式。它们需要用户预先配置系统,包括完整性规则(例如 Holistic [11])、统计参数(例如 SCARE [48])或两者(例如 HoloClean [37])。他们的想法是,根据成本函数(例如数值变化的数量)最小化数据集中的数值交换,直到数据集与完整性规则和统计可能性保持一致。在缺乏冗余数据以及正确、完整的用户预设规则和参数集来定义一致性的情况下,这些系统很难在许多数据集中同时实现高精度和高召回率。

Table 1: A dirty dataset d and its cleaned version d^*.

ID	Name	Address
1	H	5th Str
2	Hana	-
3	Gandom	7th Street
4	Chris	9th Str

ID	Name	Address
1	Hana	5th Street
2	Hana	5th Street
3	Gandom	7th Street
4	Christopher	9th Street

示例 1(现有系统的局限性)。表 1 显示了一个不完整的数据集,其中已经检测到的数据错误用红色标记。目标是修复这些数据错误并生成描述的清理后的数据集 d*。由于上述限制,上述系统将无法有效地修复这些数据错误。首先,由于数据集没有很高的冗余度,这些系统无法找到数据错误“9th Str”的更正,因为正确的值“9th Street”没有出现在数据集中的任何地方。其次,我们没有可以定义这个数据集的一般完整性规则。功能依赖关系 *Name→Address* 仍然不足以修复数据错误“5th Str”,因为元组 1 和 2 在名称列中的值不同。

3. Baran 概述

图 1 展示了 Baran 的工作流程。给定一个带有标记数据错误的脏数据集和一个可选的修订数据集,Baran 借助用户反馈修正数据错误,并返回数据集的清理版本。该工作流程包括一个在线阶段和一个可选的离线阶段。

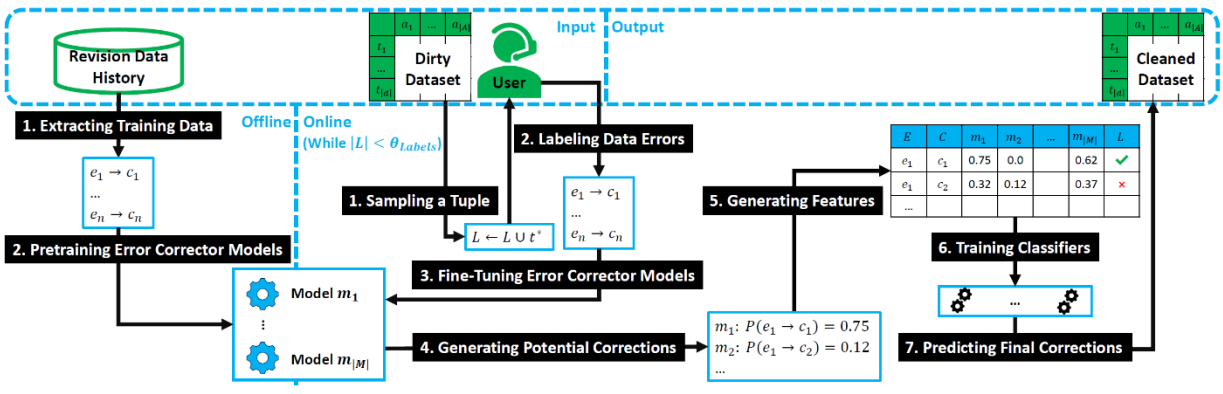


Figure 1: The workflow of Baran.

3.1 在线阶段

在线阶段旨在对当前数据集上的错误校正器模型进行微调,并组合它们以修复数据错误。如果错误校正器模型已经预先训练过,Baran 会对其进行增量更新。否则,Baran 会从当前数据集上从头开始训练错误校正器模型。Baran 在在线阶段的每次迭代中执行以下步骤。

步骤 1 和 2:对元组进行采样和标记。在每次迭代中,Baran 都会对一个元组进行采样,由用户对其进行标记。因此,迭代的总次数受用户标记预算的限制。然后,Baran 要求用户修复采样元组中已标记的数据错误。为了充分利用有限的用户标记,采样元组集应尽可能多地覆盖数据集中的数据错误类型。我们将在第 4.2.1 节中详细介绍这些步骤。

步骤 3:微调误差校正器模型。Baran 根据用户更正的数据错误更新误差校正器模型。在第 4.1 节中,我们为所有不同的误差校正器模型定义了一个统一的模型,以便在每次出现新的用户更正的数据错误时,以相同的方式逐步更新所有模型。

步骤 4 和 5:生成修正候选和特征。每个错误修正模型为每个数据错误提出不同的修正候选。我们需要表示每个特定修正候选对每个特定数据错误的适应性。正如第 4.2.2 节中详细描述的那样,Baran 生成一个特征向量,表示数据错误和修正候选的相互适应性。

步骤 6 和 7:训练和应用分类器。由于用户修复了抽样元组中的数据错误,因此我们获得了对一小部分数据错误的实际修正。因此,我们可以设计一个二元分类任务,由分类器来判断修正候选是否是数据错误的实际修正。在第 4.2.2 节中,我们将讨论 Baran 如何根据特征向量和用户标签来训练每列的二元分类器。Baran 应用经过训练的分类器来预测其余数据错误的最终校正。

3.2 离线阶段

在可选的离线阶段,Baran 通过处理任何提供基于值的校正的外部源来预训练基于值的错误校正器模型。

步骤 1:提取训练数据。基于值校正的外部数据集可能并不完全结构化。这就是我们为什么需要首先从非结构化/半结构化的外部源中提取基于值的校正,正如第 5 节所述。

步骤 2:预训练误差校正器模型。Baran 根据提取的训练数据预训练基于值的误差校正

器模型。我们将在第 5 节详细介绍此步骤。

4. 误差校正引擎

我们首先介绍误差校正器模型以及如何对其进行微调。然后,我们将解释如何整合这些模型的输出并修复数据错误。

4.1 误差校正器模型

纠错模型是一种算法,它可以使用错误数据的上下文逻辑对数据进行修正。纠错模型集最好能够包含完整、正确且自动的纠错器。事实上,它们的组合应该能够准确(正确)地修复所有数据错误(完整性),且无需用户参与(自动化)。同时满足这三个要求来选择纠错模型并非易事,因为通常需要权衡其中的利弊。因此,在设计纠错模型时,我们必须权衡正确性、完整性和自动化的关系。

我们首先忽略错误校正模型的精确度,以此来解决这种权衡问题。我们的错误校正模型旨在首先提出尽可能多的潜在校正。我们通过自动生成一组校正候选,提高可实现的召回率。这些校正候选中可能有许多是不相关的。不过,在使用这些自动提出的校正候选训练分类器时,我们会在稍后避免误报。

鉴于上述思路,我们设计了一套纠错器模型,每个模型都利用启发式数据错误的一种情境。为了使纠错模型简单、通用且可逐步更新,我们将纠错模型 m 正式定义为条件概率,其中 e_m 是模型 m 处理数据错误 e 所使用的上下文;count(e_m)是数据错误上下文 e_m（在数据集的不同位置）被观察到的次数;count(c|e_m)是上下文 e_m 被用于将数据错误 e 修正为值 c 的次数。

$$P(c|e_m) = \frac{\text{count}(c|e_m)}{\text{count}(e_m)}, \qquad (1)$$

直觉是,数据错误上下文 e_m 用于将数据错误 e 修正为更正 c 的次数越多,更正 c 就越可能成为数据错误 e 的实际更正。请注意,我们可以通过为每对数据错误上下文 e_m 和更正 c 存储 count(c|e_m)和 count(e_m)来逐步更新此模型。这种误差校正器模型的定义是抽象的,需要针对每种模型类型进行相应的实现。

虽然我们提出了一套默认且通用的纠错器模型,适用于各种数据集,但用户也可以使用可选的自定义纠错器模型来扩展模型集。特别是,用户可以选择以纠错器模型的形式实现

数据约束,并将其整合到 Baran 中。由于 Baran 将纠错器模型视为黑匣子,因此它概括了之前的纠错聚合器[35, 37]。

4.1.1 基于值的模型

基于值的纠错器模型仅使用错误的值本身来学习修复数据错误[19]。在这里,数据错误 e 及其上下文 e_{val} = d[i,j]是相同的。每当错误的值 e_{val} 被校正为值 e_{val}^*时,Baran 通过编码错误的值及其校正操作来更新基于值的纠错器模型。

错误值编码。提取错误值是训练基于值的错误校正器模型的重要技术。数据值的编码有不同的抽象级别,例如提取字符类别或字符串长度。我们利用两个简单而通用的编码器。第一个编码器是同一性编码器,它用原始字符对值进行编码。这种编码适用于修复语义数据错误。例如,要将错误的国名“Holland”更正为“Netherlands”,我们的基于值的错误校正模型需要看到确切的错误值“Holland”。我们的第二个编码器是 Unicode 编码器,它使用等效的 Unicode 类别[47]对数据值的每个字符进行编码。例如,大写字符将替换为其类别符号“<Lu>”,数字将替换为其类别符号“<Nd>”。这种编码方式通过概括数据错误的语法,使基于值的模型能够更快地学习句法错误校正。

示例 2(错误值编码)。考虑错误值 e_{val} =“16/11/1990”和更正后的值 e_{val}^* =“16.11.1990”,我们有两种方法来编码这个错误值。同一性编码器用原始字符编码错误值: 当错误值等于“16/11/1990”时,可能的更正操作是将“/”替换为“.”。另一方面,Unicode 编码器通过将字符抽象为 Unicode 类别来编码这个错误值:每当错误值采用“<Nd><Nd><Po><Nd><Nd <Po><Nd><Nd><Nd><Nd>”格式时,可能的更正操作是将“/”替换为“.”。虽然第一种编码非常准确,但第二种编码可以提高整体召回率。

更正操作编码。对错误值进行编码后,我们需要对所需的更正操作进行编码。一般来说，能应用于任何错误值的更正操作有四种：

1. *Remover operator* (移除操作符)。该操作符用于移除子串。例如,如果要将错误值“U.S.”修正为“US”,移除操作符可以移除“.”。

2. *Adder operator* (添加操作符)。添加操作符用于添加子串。例如,如果要将错误值“US”修正为“U.S.”,添加操作符可以添加“.”。

3. *Replacer operator* (替换操作符)。替换操作符可以同时删除和添加子字符串。例如,如

果我们要将错误值“16/11/1990”修正为“16.11.1990”,替换操作符可以删除“/”,并添加“.”。

4. *Swapper operator* (交换操作符)。交换操作符用另一个值代替整个错误值。虽然之前的基于值的操作符适用于语法错误,但交换操作符可用于修复语义数据错误。例如,如果我们要将错误值“Holland”修正为“Netherlands”,交换操作符可以用来代替这些值。

我们训练基于价值的模型,每个模型都学习对编码错误值执行的一种校正操作。通过这一组操作符,我们可以生成任何基于值的校正。给定用户校正的数据错误,Baran 根据差异检查技术[22]计算错误值 e_{val} 和校正值 e_{val}^*在字符级别的差异,并提取操作符。

示例 3(差异检查)。考虑错误值 e_{val}=“Chris Edward NoLan”和用户更正值 e_{val}^*=“Christopher Nolan”，分别作为源字符序列和目标字符序列，差异检查算法的输出如下:

$$\text{diff}(e_{val}, e_{val}^*) = \begin{cases} \texttt{Add ``topher'' after ``Chris''.} \\ \texttt{Remove ``Edward ''.} \\ \texttt{Replace ``L'' with ``l''.} \end{cases}$$

因此,我们可以用这个用户更正的例子来更新四个运算符的基于值的模型。

总的来说,我们有 2 个错误值编码器和 4 个校正操作符,故我们有 2×4=8 个基于值的纠错器模型。为了实现公式 1 中错误校正器模型的抽象定义,基于值的模型将编码的错误值 encode(e_{val})视为上下文 e_m，并将必须应用于该错误值的校正操作 o 视为校正 c。

形式地，$P(c|e_m) = P(o|\text{encode}(e_{val})) = \frac{\text{count}(o|\text{encode}(e_{val}))}{\text{count}(\text{encode}(e_{val}))}$.

示例 4 (基于值的模型)。假设我们有一个基于值的错误校正模型,它配备了同一性编码器和交换操作符。假设该模型在数据集的不同位置 5 次遇到错误值“Holland”,即 count(encode(e_{val})) = 5。假设用户将这个数据错误修正为“Netherlands”4 次,修正为“HL”1 次。我们分别称这两个交换操作为 o_1 和 o_2。因此,这个基于值的错误校正模型为任何检测到值为“Holland”的数据错误 e 提出了两个校正候选:“Netherlands”(P(o_1|encode(e_{val})) = 0.8)和“HL”(P(o_2|encode(e_{val})) = 0.2)。

4.1.2 基于邻域的模型

基于邻域的错误校正模型根据列之间的关系来学习修复数据错误。基于邻域的模型根据活跃域与其他列干净数据值之间的关系,提出了将活动域的干净值作为潜在的修正。我们将各种列的关系和关联[4]限制为左侧有一个属性的函数依赖关系。这样,我们就可

以合理地限制所有函数依赖的指数空间,因为这些函数依赖对于数据清理更有用[32]。与错误检测系统 Raha [27]类似,我们认为对于$\forall j_1 \neq j_2 \in [1,|A|]$, $j_1 \rightarrow j_2$ 都是功能依赖关系。为了实现公式 1 中错误校正器模型的抽象定义,对于每个功能依赖关系 $j_1 \rightarrow j_2$,基于邻域的模型将邻近上下文 e_{vic} = d[i,:]中的干净共现值 d[i,j_1]视为上下文 e_m,将干净值 d[i,j_2]作为修正 c。形式上,$P(c|e_m) = P(d[i,j_2]|d[i,j_1]) = \frac{count(d[i,j2]|d[i,j1])}{count(d[i,j2])}$. 这个条件概率显示了左侧值 d[i,$j_1$]决定右侧值 d[i,$j_2$]的频率。

总的来说,我们有|A|×(|A|-1)个基于邻域的错误校正模型,因为我们考虑了每个属性的功能依赖性。

示例 5 (基于邻域的模型)。考虑示例 1 中 *Name*→*Address* 的功能依赖性,基于邻域的模型得知名称值“Gandom”必须始终具有地址值“7th Street”。因此,对于 *Address* 列中任何具有邻列 *Name* 值为 “Gandom”的数据错误 e,对应的基于邻近的模型会提出一个修正候选 P(“7th Street”|“Gandom”)=1.0。

4.1.3 基于领域的模型

基于领域的错误校正器模型利用其所在列中的现有值来学习并修复数据错误。基于领域的模型从活动域中提出最相关的干净值作为潜在的校正。由于数据集中的元组通常彼此独立,因此列中值的顺序、距离或邻近并不表示任何相关性。然而,干净值的频率可以用作估计其相关性的信号。同一列中更频繁的干净值更有可能是同一列中数据错误的更正。因此,为了实现公式 1 中错误校正器模型的抽象定义,基于领域的模型将域上下文 edom = d[:,j]视为上下文 e_m,并将该领域内的每个干净值视为更正 c。形式上,$P(c|e_m) = P(d[i,j]|e_{dom}) = \frac{count(d[i,j]|edom)}{count(edom)}$, 其中 count($e_{dom}$)是干净值的数量,count(d[i,j]|$e_{dom}$)是干净值 d[i,j]在数据误差的有效域中的频率。这个条件概率显示了在 j 列中观察到干净值 d[i,j]的概率。

总体而言,我们为每列建立了一个基于领域的错误校正模型,即|A|个基于域的模型。

示例 6 (基于领域的模型)。考虑例 1 中 *Name* 列的干净值,基于领域的模型得知,所有干净值都有相同的概率来校正该列中的每个数据错误,因为它们都只出现一次。因此,相应的基于领域的模型为 *Name* 列中的任何数据错误 e 提出了两个修正候选 P(“Hana”|Name) = 0.5 和 P(“Gandom”|Name) = 0.5。

4.2 学习集成模型

训练过的错误校正模型利用其值、邻域和领域上下文为任何数据错误生成各种可能的校正。因此,我们需要从不同模型的所有拟校正候选中确定实际的校正。首先,我们需要定义适当的采样和标记策略来收集用户标记。然后,我们需要设计一个特征表示和一个分类任务,以便能够有效且高效地评估所有数据错误的所有生成的校正。

4.2.1 元组抽样和标记

Baran 以对数据错误示例进行有限次数的手动更正的形式,将用户监督纳入其中。它利用这些示例来更新所有错误校正模型并训练分类器。为了对元组进行用户标记抽样,Baran 遵循一种迭代过程。在每次迭代中,Baran 都会绘制一个元组 t,使元组评分公式

$$t^* = \underset{t \in d}{\operatorname{argmax}} \prod_{d[i,j] \in t \cap E'_j} \exp\left(\frac{|E'_j|}{|E_j|}\right) \exp\left(\frac{\operatorname{count}(d[i,j]|E'_j)}{|E'_j|}\right), \quad (2)$$

最大化。其中,E_j 是第 j 列中的所有数据错误的集合;E'_j 是第 j 列中所有尚未修复的数据错误集合;$\text{count}(d[i,j]|E'_j)$是第 j 列中未修复的且值恰好为 d[i,j]的数据错误数量。这个评分公式适用于以下情况:(1)包含更多未修复的数据错误;(2)数据错误存在于有大量未修复错误的列中;(3)错误值在未修复的数据中频繁出现。这样,Baran 为标记不足的列的分类器获得了信息丰富的标记数据点。一旦用户修正了样本元组 t 的数据错误,基于值、基于邻域和基于领域的模型将相应更新。

示例 7 (更新模型)。假设 Baran 对示例 1 中的第一个元组进行采样。用户修正了该元组中的数据错误。因此,基于值的模型将使用新的错误修复实例进行更新，即把“5th Str”的错误值修正为“5th Street”。特别地,一个基于值的模型,使用 Unicode 编码器和加法运算符,会学习在“Str”后面添加“eet”,并把这种方法用于所有类似于“5th Str”模式的错误值。相应的基于邻域的 *Name→Address* 模型将更新为“Hana”和“5th Street”的新关联。此外,基于领域的 *Address* 列模型也将更新,因为该列中现在有 2 个干净的值。

在数据清理中,选择正确的人工监督形式一直是一个挑战。大多数数据清理任务都需要人工监督,因为所需的更正可能取决于用例或具有主观性。然而,如果人工监督本身是错误的,那么数据清理任务也将存在缺陷。在以往的基于约束的方法中,这个问题以过时的[43]或不准确的[9]完整性约束的形式出现。

在我们的示例驱动系统中,这个问题可能以错误的用户更正示例的形式出现。然而,我们认为,基于规则的算法在解决这个问题上不如基于约束的算法,原因有三。首先,基于例子的监督比规则生成更简单,因此也更不容易出错。其次,Baran 将所有人工监督手段(如提供规则、参数、外部源和注释)限制为仅标记几个元组。在现有方法中,标签的数量与数据集的大小成比例(例如,数据集的 1%到 10%[20, 41]),而在 Baran 中,标签的数量与数据集的数据错误类型成比例(即 10 到 20 个元组)。这种有限的人工交互自然也减少了人为错误的可能性。同时,用户总是可以跳过难以标记的示例。第三,在通过示例学习时,可以通过简单地考虑更多用户提供的示例来弥补人为标记错误。然而,基于约束的方法并不那么灵活,因为错误的输入规则很难通过数据集中的其他规则来弥补。

4.2.2 特征生成和分类

我们现在可以利用用户标签来训练分类器,从而预测数据错误的最终更正。一种直接的方法是定义一个多类分类任务,其中每个更正候选都类似于一个目标类,分类器必须为每个数据错误选择其中一个目标类。然而,将此分类任务定义为多类分类会导致特征向量的稀疏性问题[10]。在当前的用例中,特征向量必须为每个校正候选模型编码所有错误校正器的概率。形式上,数据错误 e 的特征向量是 $v(e) = [P(c|e_m) \mid \forall m \in M, \forall c \in C]$; 其中 M 是所有模型的集合,C 是所有校正候选的集合。因此,特征向量的规模将随着校正候选的数量而变化,但并非每个校正候选都与每个数据错误相关，结果是特征向量中会有大量的零元素。

示例 8 (多类分类)。假设我们有 20 个错误校正器模型,每个模型针对任何数据错误提出 100 个校正候选。在多类分类任务中,我们必须将所有 20×100=2000 个模型的概率编码为每个数据错误的特征向量。

为了避免多类分类的稀疏性问题,我们将分类任务定义为二元决策。二元分类器的角色是决定候选修正是否是数据错误的实际修正。我们生成一个特征向量,表示一个特定数据错误中特定修正的适应性。因此,对于数据错误 e 和候选修正 c 的任意组合,我们将所有错误修正模型的概率作为特征向量进行收集。形式上,$v(e,c) = [P(c|e_m) \mid \forall m \in M]$; (3) 其中, M 是所有错误校正模型的集合。当特征向量中包含的概率为接近 1(即对于大多数模型 $m \in M, P(c|e_m) \approx 1.0$)时,候选修正项 c 更有可能是对数据错误 e 的实际修正;因为在这种情况下,大多数具有高置信度的错误修正模型都会为该数据错误提出这个修正项。

因此,特定列 j 中数据错误的特征向量集为 $V_j = \{v(e,c) \mid \forall e \in E_j, \forall c \in C_e\}$; (4) 其中,$E_j$ 是列 j 的数据错误集,C_e 是数据错误 e 的所有修正候选集。特征向量的数量取决于错误校正器模型提出的修正候选数量。

与多类分类任务中的特征表示相比,这种特征表示具有三个优点。首先,这种特征向量小而密集。特征的数量等于错误校正器模型的数量,非零特征的比例更高,因为所考虑的模型与当前(e,c)对相关。其次,这种特征表示可以快速收敛,因为我们可以将一个用户标签转换为多个训练数据点。假设用户用校正 c^*修正了数据错误 e。Baran 将这个用户标签扩展为多个训练数据点。自然地,我们有一个正数据点,指示修正 $c^* \in C_e$ 是数据错误 e 的实际修正。此外,我们有许多负数据点,指示所有修正 $c \in C_e \backslash \{c^*\}$不是数据错误 e 的实际修正。第三,这个高度不平衡的训练集,只有少数正数据点,而负数据点数量庞大,使得我们的分类器在预测修正时变得保守。事实上,我们的分类器更倾向于负类,从而避免误报。这就是为什么我们的系统精度普遍较高的原因。

我们不对整个数据集训练一个分类器,而是对每一列训练一个二元分类器,因为数据错误、所需的校正技术以及上下文的有用性在各自领域内具有更好的可比性。尽管 Baran 对每列训练一个分类器,但它通过基于邻域的错误校正器模型保留了所有列之间的依赖信号,这些信号被编码为数据错误和校正候选的每对特征。基于邻域的模型根据给定列与不同列的功能依赖关系给出数据错误的校正。

在每次迭代中,Baran 都会训练所有分类器,并将每个分类器应用于相应列的所有数据错误。分类器不会覆盖用户提供的更正。对于每对数据错误 e 和更正候选 c,相应的分类器会预测一个带有置信度的标签。对于数据错误 e,分类器可以确定零个或多个更正候选 c 作为最终更正。如果二元分类器对每个修正候选预测的标签都是 0,则不会对相应数据错误进行任何修正。如果它对多个修正候选预测的标签都是 1,则 Baran 会选择置信度最高的修正作为最终修正。只要 $|L| < \theta_{Labels}$,就会重复这一迭代过程,其中 $|L|$ 是标记元组的数量。Baran 将最后一次迭代的输出视为最终系统输出。

示例 9 (特征生成和分类)。考虑示例 1 中的 *Address* 列,下表包含数据错误和校正候选及其它们对应的特征。简洁起见,我们只展示了几个(e,c)对和几个特征。这些特征包括基于值的模型(Unicode 编码器和加法运算符($m_{Unicode+Adder}$))、基于邻域的模型($m_{Name \rightarrow Address}$)和基于领域的模型($m_{Address}$)。用户在示例 7 中已验证了微型数据集的元组 1，所

以我们将该数据集的前两对做了标记。

Error	Correction	$m_{Unicode+Adder}$	$m_{Name \rightarrow Address}$	$m_{Address}$	Label
5th Str	5th Street	1.0**	1.0*	0.5	1
5th Str	7th Street	0.0	0.0	0.5	0
-	5th Street	0.0	1.0*	0.5	
-	7th Street	0.0	0.0	0.5	
9th Str	5th Street	0.0	0.0	0.5	
9th Str	7th Street	0.0	0.0	0.5	
9th Str	9th Street	1.0**	0.0	0.0	

Address 列的分类器将这些对作为数据点接收。它使用前两个标记的数据点进行训练,然后预测其余未标记数据点的标签。特别地, 根据基于邻域的特征 $m_{Name \rightarrow Address}$ (用*标记),分类器预测“5th Street”为错误值“-”的最终更正。该模型返回了 1.0 的概率(“-”和“5th Street”),因为在给例 7 中的元组 1 标注并更新模型后,基于邻域的模型学习了 *Name* 列中的值“Hana”和 *Address* 列中的值“5th Street”之间的关联。此外,分类器预测“9th Street”为错误值“9th Str”的最终修正,因为基于值的特征 $m_{Unicode+Adder}$ (标记为**)。该模型对(“9th Str”、“9th Street”)的返回概率是 1.0,因为根据 Unicode 编码,“5th Str”的数据错误与“9th Str”匹配。修正候选项“9th Street”是通过与“5th Str”生成“5th Street”相同的加法运算符而生成的。

5.预训练模型

到目前为止,校正候选要么由给定数据集中的正确值提供,要么通过用户校正提供。这种方法可能面临两个普遍的限制。首先,用户校正的数量有限,可能不足以充分训练错误校正器模型。其次,一些数据集外的校正可能永远无法找到。幸运的是,我们的问题表述允许我们从外部来源提取额外的校正候选,以预训练错误校正器模型。

如前所述,我们有三组错误校正器模型。由于基于邻域和基于领域的错误校正器模型与模式相关,因此应在具有相同模式的数据集上进行预训练。因此,预训练它们将非常简单,因为历史数据将是具有相同模式的结构化数据集。基于值的错误校正器模型可以独立于模式,因此可以在任何可以提取值校正的数据集上进行预训练。在没有结构化数据集的情况下,我们可以使用非结构化/半结构化数据集,其中包含用户提供的更正。有一些公开可用的通用修订历史,例如维基百科页面修订历史。从这些修订历史中提取基于价值的更正需要两个主要步骤。首先,我们需要将非结构化/半结构化修订文本分解为文本片段(即块)。其次,我们需要对后续修订中的文本片段进行排列,以收集基于价值的更正。在这里,我们简要讨论了如何将这些步骤应用于维基百科页面修订历史,将其作为通

用且公开的修订数据集。要将相同的方法应用于其他类似的修订历史(例如网页),我们只需要将文本分段器调整为相应的标记语言,例如 HTML。

维基百科页面修订历史包含人类提交的修订内容,容量达数 TB。这些内容可在维基百科发布的转储文件中找到[1]。半结构化的维基百科页面使用维基文本标记语言编写,该语言可识别一组实体[2]。为了分割文本,Baran 将每个页面修订文本递归分解为各个实体。然后,Baran 通过再次执行差异检查,但这次是在段级,将每个连续段列表中的相应段对齐。Baran 舍弃了涉及空值的基于值的更正。最后,Baran 预训练了更正示例。预训练的模型给当前数据集的数据错误生成更多的更正候选。因此,Baran 可以在相同的用户标记预算下更正更多的数据错误,因为现在有更少的更正候选,同时,预训练模型的相应特征为分类器提供了更多的证据。请注意,Baran 仍然可以在用户标签的帮助下避免不相关的修正候选。用户标签让分类器在所有可用的修正候选中学习并确定优先级。

示例 10 (预训练模型)。维基百科页面 P = {r_1,r_2}经过两次修订,其历史如下:

$$r_1 = \text{“’’’Chris Nolan’’’ (born on ’’30/07/1970’’) is a well-known [[British]] film-maker.”}$$
$$r_2 = \text{“’’’Christopher Nolan’’’ (born on ’’30.07.1970’’) is a well-known [[English]] filmmaker.”}$$

Baran 将这些页面修订文本分解为两个文本片段列表:

$$S_1 = [\text{“Chris Nolan”, “(born on ”, “30/07/1970”, “) is a well-known ”, “British”, “ film-maker.”}]$$
$$S_2 = [\text{“Christopher Nolan”, “(born on ”, “30.07.1970”, “) is a well-known ”, “English”, “ filmmaker.”}]$$

然后,Baran 将两个连续片段列表的相应部分对齐:

$$\text{diff}(S_1, S_2) = \begin{cases} \text{Replace ‘‘Chris Nolan’’ with ‘‘Christopher Nolan’’.} \\ \text{Replace ‘‘30/07/1970’’ with ‘‘30.07.1970’’.} \\ \text{Replace ‘‘British’’ with ‘‘English’’.} \\ \text{Replace ‘‘ film-maker.’’ with ‘‘ filmmaker.’’.} \end{cases}$$

这样,我们就可以获得更多基于值的更正作为训练数据,例如将“Chris Nolan”更正为“Christopher Nolan”。我们还对每个片段进行标记,并增加基于值的更精细的更正示例,例如将“Chris”更正为“Christopher”。

用这些新的校正示例对基于值的模型进行预训练,可以修正示例 1 中微型数据集的最后一个剩余数据错误,而无需任何进一步的用户标记。考虑示例 1 中的 *Name* 列,下表包含

数据错误和修正候选及其对应特征的配对。为简洁起见,我们仅展示几对和几个特征。这些特征是一个基于值的模型,包含同一性编码器和加法运算符($m_{Identity+Adder}$)和一个基于领域的模型(m_{Name})。前两对被标记为我们在示例 7 中微型数据集中的用户已验证元组 1。

Error	Correction	$m_{Identity+Adder}$	m_{Name}	Label
H	Hana	1.0*	0.67	1
H	Gandom	0.0	0.33	0
Chris	Hana	0.0	0.67	
Chris	Gandom	0.0	0.33	

Name 列的分类器将这些数据对作为数据点。它使用前两个标记的数据点进行训练,然后预测其余未标记数据点的标签。仅凭这两个标记的数据点,分类器无法将错误值“Chris”修正为实际的正确值“Christopher”,因为它根本不在候选修正值中。然而,如果我们用之前从维基百科中提取的“Chris”到“Christopher”的修正示例来预训练基于值的模型 $m_{Identity+Adder}$,我们将得到以下新的数据错误和修正候选对作为新的数据点。

Error	Correction	$m_{Identity+Adder}$	m_{Name}	Label
Chris	Christopher	1.0*	0.0	

分类器现在已有足够证据修正错误值“Chris”,无需用户进一步标注。由于用户已经将错误值“H”修正为“Hana”,分类器了解到基于值的模型 $m_{Identity+Adder}$(标记为*)是一个重要特征。分类器还观察到,基于值的特征 $m_{Identity+Adder}$ 对(“Chris”,“Christopher”)具有很高的概率,因为模型在预训练阶段已经学习了这种概率。因此,分类器可以预测“Christopher”是错误值“Chris”的最终修正。

6. 实验

我们的实验旨在回答以下问题。(1) 具有和不具有迁移学习的 Baran 与现有的纠错系统相比如何? (2) 纠错器模型如何影响系统性能? (3) 我们的元组采样方法的影响是什么? (4) 分类器的选择如何影响系统性能? (5) 上游错误检测引擎的结果质量如何影响纠错性能? 我们首先介绍实验设置,然后详细介绍实验。

6.1 设置

数据集。我们使用表 2 中列出的 7 个现有文献中的知名数据集来评估我们的系统。纠错任务的难度取决于错误率、错误类型的多样性以及错误上下文信号的可用性。我们

手动检查了数据集,以确定普遍的数据错误类型以及用于纠正这些数据错误的有用上下文信息。

Table 2: Dataset characteristics. The error types are missing value (MV), typo (T), formatting issue (FI), and violated attribute dependency (VAD) [36].

Name	Size	Error Rate	Error Types	Data Constraints
Hospital	1000 × 20	3%	T, VAD	city → zip, city → county, zip → city, zip → state, zip → county, county → state, index (digits), provider number (digits), zip (5 digits), state (2 letters), phone (digits)
Flights	2376 × 7	30%	MV, FI, VAD	flight → actual departure time, flight → actual arrival time, flight → scheduled departure time, flight → scheduled arrival time
Address	94306 × 12	14%	MV, FI, VAD	address → state, address → zip, zip → state, state (2 letters), zip (digits), ssn (digits)
Beers	2410 × 11	16%	MV, FI, VAD	brewery id → brewery name, brewery id → city, brewery id → state, brewery id (digits), state (2 letters)
Rayyan	1000 × 11	9%	MV, T, FI, VAD	journal abbreviation → journal title, journal abbreviation → journal issn, journal issn → journal title, authors list (not null), article pagination (not null), journal abbreviation (not null), article title (not null), article language (not null), journal title (not null), journal issn (not null), article journal issue (not null), article journal volume (not null), journal created at (date)
IT	2262 × 61	20%	MV, FI	support level (not null), app status (not null), curr status (not null), tower (not null), end users (not null), account manager (not null), decomm dt (not null), decomm start (not null), decomm end (not null), end users (not 0), retirement (predefined list), emp dta (predefined list), retire plan (predefined list), division (predefined list), bus import (predefined list)
Tax	200000 × 15	4%	T, FI, VAD	zip → city, zip → state, first name → gender, area code → state, gender (predefined list), area code (3 digits), phone (7 formatted digits), state (2 letters), zip (non-zero-leading digits), material status (predefined list), has child (predefined list), salary (digits)

Hospital[37]和 *Flights*[25, 37]具有丰富的上下文信息,包括以重复元组和相关列形式存在的大量数据冗余。同时,由于不同原因,它们也是具有挑战性的数据集。由于 *Hospital* 的错误数据很少且随机产生,因此在这个数据集上,对信息元组进行采样具有特别大的挑战性。另一方面,由于 *Flights* 的错误率很高,因此可信的上下文信息较少。*Address* 是专有的,而 *Tax* 是 BART 数据库中的合成数据集[7]。这两个数据集都很大,包含各种数据错误类型,但重复元组和相关列较少。它们的大容量为查找实际更正带来了巨大的搜索空间。*Beers* [21]、*Rayyan* [30]和 *IT* [3]也是由数据集所有者清理的真实数据集。这三个数据集缺乏数据冗余,这使得修复数据错误变得具有挑战性。

基准。我们将 Baran 与 4 个最新的基准系统进行了比较。

- KATARA [12]是一个由知识库驱动的数据清理系统,它以一组实体关系作为输入,并相应地修复违反规则的数据错误。我们使用 DBpedia 知识库[8]中可用的所有实体关系运行了 KATARA。

- SCARE [48]是一个错误校正系统,它对数据集进行分区,并根据统计可能性使用清洁值选择数据错误的校正。我们使用随机数据分区运行了 SCARE。我们将其最大值校正数设置为检测到的错误数(δ= |E|),以适应所有数据错误。

- Holistic [11]是一个使用拒绝约束的数据清理系统。我们使用数据集所有者提供的所有数据约束(即完整性规则和列模式)运行了 Holistic(表 2)。

- HoloClean [37]是一个纠错系统,利用完整性规则、匹配依赖关系和统计信号来整体修复数据错误。我们使用数据集所有者提供的所有数据约束和匹配依赖关系运行了 HoloClean(表 2)。

评估指标。我们报告了精确度、召回率和 F_1 评分,以评估有效性。精确度是指正确修复的数据错误数量除以所有修复的数据错误数量。召回率是指正确修复的数据错误数量除以所有数据错误数量。F_1 评分是精确度和召回率的调和平均值。我们还报告了运行时间(以秒为单位)。我们报告了标记元组的数量,以评估人工参与度。对于每个指标,我们报告 10 次独立运行的平均值。

Baran 默认设置。默认情况下,我们使用所有描述的错误校正器模型运行 Baran,而不对其进行预训练。我们使用 AdaBoost [16]作为分类器,并将标签预算设置为 θ_{Labels}=20。我们在 Ubuntu 16.04 LTS 机器上运行实验,该机器具有 28 个 2.60 GHz 内核和 264 GB 内存。Baran 与我们的错误检测系统 Raha 集成在一起,并且可以在线使用。

6.2 与基准的比较

我们将 Baran 的性能与基准的性能在有效性、效率和人工参与度方面进行比较。本节中的所有错误校正系统都以相同正确且完整的数据错误集作为输入。

Table 3: System effectiveness in comparison to the baselines.

System	Hospital			Flights			Address			Beers			Rayyan			IT			Tax		
	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F
KATARA	0.98	0.24	0.39	0.00	0.00	0.00	0.79	0.01	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.04	0.01	0.02	0.59	0.01	0.02
SCARE	0.67	0.53	0.59	0.57	0.06	0.11	0.10	0.10	0.10	0.16	0.07	0.10	0.00	0.00	0.00	0.20	0.10	0.13	0.01	0.01	0.01
Holistic	0.52	0.38	0.44	0.21	0.01	0.02	0.41	0.31	0.35	0.49	0.01	0.02	0.85	0.07	0.13	1.00	0.78	0.88	0.96	0.26	0.41
HoloClean	1.00	0.71	0.83	0.89	0.67	0.76	0.01	0.01	0.01	0.01	0.01	0.01	0.00	0.00	0.00	0.01	0.01	0.01	0.11	0.11	0.11
Baran	0.88	0.86	0.87	1.00	1.00	**1.00**	0.67	0.32	**0.43**	0.91	0.89	**0.90**	0.76	0.40	0.52	0.98	0.98	**0.98**	0.84	0.78	0.81
Baran (with TL)	0.94	0.88	**0.91**	1.00	1.00	1.00	0.67	0.32	0.43	0.94	0.87	0.90	0.80	0.44	**0.57**	0.98	0.98	0.98	0.95	0.73	**0.83**

有效性。表 3 显示了所有系统在错误校正方面的有效性。在所有数据集的 F_1 评分方面,Baran 都优于所有基准,因为我们的错误校正方法实现了高精度和高召回率。事实上,由于 Baran 根据数据错误的不同背景训练了一组全面的错误校正器模型,这些模型提出了大量潜在的校正,从而显著提高了可实现的召回率。之后,当 Baran 利用少量用户标签将这些潜在更正整合为最终更正时,也能保持高纠错精度。

Baran 的有效性取决于每个数据集提供的基于值、基于邻域和基于领域的上下文信息量。在信息丰富的数据集(如 *Hospital* 和 *Flights*)上,由于存在许多重复行和相关列,Baran 可以利用所有三个数据错误上下文来训练和集成有效的错误校正器模型。因此,Baran 在这些数据集上获得了较高的 F_1 分数。特别是,Baran 在 *Flights* 数据集上表现完美,该数据集具有高度冗余性。另一方面,在错误上下文信息较少的数据集上,例如 *Address*,其中错误值通常是缺失值,Baran 无法准确修复所有数据错误。

在超过 300000 个维基百科页面的修订历史(表 3 中的 *Baran*(带 *TL*)行)上预训练基于值的模型,可以显著提高存在普遍句法数据问题的数据集(如 *Hospital* 和 *Rayyan*)的 F_1 评

分。预训练为分类器提供了更多的证据来预测实际更正。特别地,预训练的基于值的模型会产生额外的校正候选,使 Baran 能够收敛到其报告的 F_1 分数,而相同的 F_1 分数所需的标记元组少于 20 个。在 *Hospital* 数据集上，预训练产生了 455390 个新的校正候选。因此,我们只需要 13 个标记元组就能达到 20 个标记元组所达到的相同的 F_1 分数。在 *Rayyan* 数据集上，预训练生成 77569 个新的校正候选。通过预训练,我们只需要 18 个标记的元组,就能达到之前需要 20 个标记的元组才能达到的 F_1 分数。在 *Tax* 数据集上,预训练生成 7134254 个新的校正候选。因此,我们只需要 19 个标记的元组,就能达到 20 个标记的元组才能达到的 F_1 分数。相反,在 *IT* 等数据集上,有效性提升幅度较小,因为其中大部分错误值都是缺失值。在这种情况下,基于值的模型无法有效地提出可能的更正,无论是否经过预训练。

KATARA 的精确度较低,因为概念的模糊性导致数据集与知识库之间不匹配。该系统的召回率也很低,因为数据集的大部分内容无法与知识库匹配。Holistic 的精确度和召回率较低,因为提供的完整性规则只能修复部分数据错误。尽管 HoloClean 在冗余度较高的数据集(如 *Hospital* 和 *Flights*)上具有相对较高的精确度和召回率,但它无法在其余数据集上达到同样的效果。在冗余度较低或预定义数据约束较少的数据集上,HoloClean 无法准确地找到数据错误的纠正方法,因为实际纠正方法要么在数据中不存在,要么不在数据约束范围内。SCARE 与 HoloClean 具有相同的缺点。

只要数据集提供丰富的上下文信息,错误率就不会显著影响 Baran 的有效性。例如,Baran 在上下文丰富的数据集 *Hospital* 和 *Flights* 上均取得了很高的有效性,尽管前者的数据错误率较低(3%),而后者则较高(30%)。为了进一步分析数据错误率的影响,我们选择了错误率最高的 *Flights* 数据集,并分别生成四个错误率为 10%、30%、50%和 70% 的无替换样本。图 2 显示了这四个数据集上错误校正系统的 F_1 评分。随着错误率的增加,所有系统的 F_1 评分都会自然下降,因为可信证据会减少。然而,Baran 始终优于所有其他系统,因为它更有效地利用了数据错误中剩余的稀缺可信上下文。

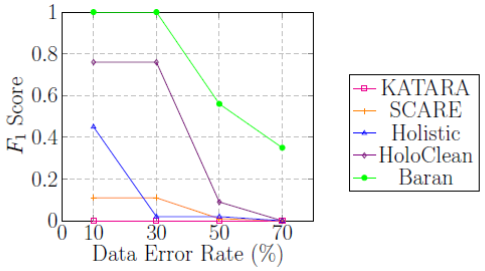


Figure 2: Scaling the error rate on *Flights*.

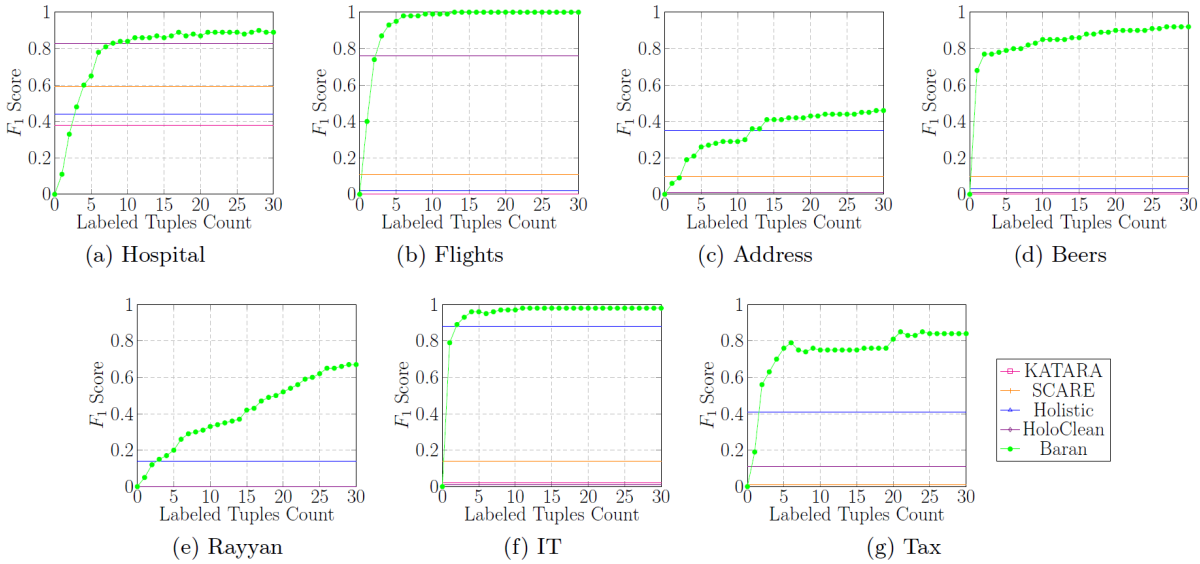


Figure 3: System effectiveness with respect to the number of labeled tuples.

人工参与。图 3 显示了系统在错误修正方面的有效性,与标记元组的数量有关。Baran 的 F_1 得分在标记元组数量较少时迅速收敛,并优于所有其他系统的 F_1 得分。这种快速收敛速度归因于 Baran 的元组采样和特征生成方法,该方法使用少量用户标记生成大量信息丰富的训练数据点。由于 KATARA、SCARE、Holistic 和 HoloClean 不使用用户标签,因此它们的 F_1 分数与标记元组的数量无关。然而,我们认为它们以其他更繁琐的形式利用了人工监督。KATARA 需要用户提供相关的知识库。SCARE 需要用户提供统计参数。Holistic 和 HoloClean 需要用户提供正确且完整的完整性规则集和匹配依赖关系。

Table 4: System runtime (in seconds).

System	Hospital	Flights	Address	Beers	Rayyan	IT	Tax
KATARA	234	116	5739	180	134	2031	15992
SCARE	76	123	11853	363	216	8717	55495
Holistic	15	10	69	9	8	3	247
HoloClean	148	39	17582	96	112	885	25778
Baran	23	22	11073	114	26	247	11936

效率。表 4 显示了系统的运行时间(以秒为单位)。虽然效率不是 Baran 的主要关注点,但与其他基准相比,它的运行时间具有竞争力。报告的运行时间捕获了 Baran 的在线阶段,因为离线阶段完全独立于输入数据集。优化机器运行时间效率并不是数据清理系统的主要目标,因为优化有效性和人工参与才是更重要的目标[3,37]。然而,开发能够在合理运行时间内工作的系统非常重要。

6.3 错误校正模型影响分析

我们对错误校正模型进行消融实验,以更好地了解它们对 Baran 整体效果的影响。首先,我们使用所有默认的错误校正模型(表 5 中的 *All* 所在行)运行 Baran。然后,我们逐个排

除每种模型,以分析其影响。例如,*All - VaM* 表示 Baran 利用了所有模型,但基于值的模型除外。最后,我们还评估了 Baran 在所有默认模型以及从表 2 中的数据约束获得的特定于数据集的自定义模型(行 *All + CM*)上的性能。Baran 将这些数据约束作为硬编码校正器,在必要时覆盖我们的默认模型。

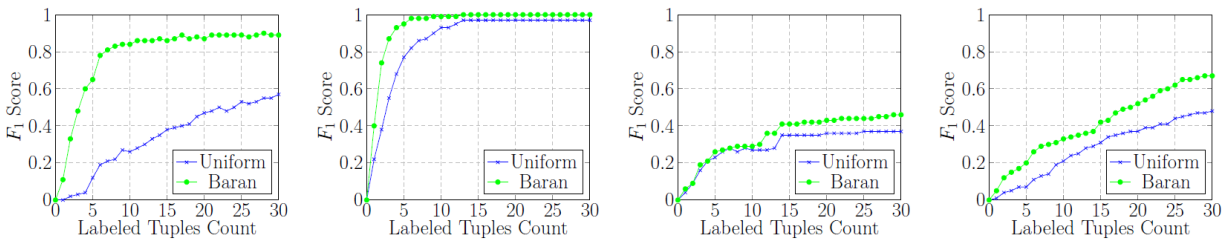
Table 5: System effectiveness with different error corrector models: value-based models (VaM), vicinity-based models (ViM), domain-based models (DoM), all default models (All), and custom models (CM).

Error Corrector Models	Hospital			Flights			Address			Beers			Rayyan			IT			Tax		
	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F
All - VaM	0.95	0.88	**0.91**	1.00	1.00	1.00	0.61	0.06	0.11	0.67	0.42	0.52	0.19	0.07	0.10	0.98	0.95	0.96	0.44	0.24	0.31
All - ViM	0.64	0.31	0.42	0.08	0.06	0.07	0.40	0.25	0.31	0.87	0.86	0.86	0.48	0.34	0.40	0.98	0.98	0.98	0.88	0.88	**0.88**
All - DoM	0.88	0.87	0.87	1.00	1.00	1.00	0.56	0.25	0.35	0.91	0.88	0.89	0.54	0.35	0.42	0.98	0.98	0.98	0.90	0.81	0.85
All	0.88	0.86	0.87	1.00	1.00	**1.00**	0.67	0.32	**0.43**	0.91	0.89	**0.90**	0.76	0.40	**0.52**	0.98	0.98	**0.98**	0.84	0.78	0.81
All + CM	0.90	0.89	0.89	1.00	1.00	1.00	0.67	0.32	0.43	0.91	0.89	0.90	0.76	0.40	0.52	0.98	0.98	0.98	0.84	0.78	0.81

如表 5 所示,在大多数数据集上,Baran 在所有默认错误校正器模型中的 F_1 得分最高。通过收集所有错误校正器模型提出的潜在校正,Baran 拥有更多的上下文信息来修复数据错误。然而,在某些数据集上,我们发现排除一种类型的错误校正器模型可以提高 F_1 得分。排除基于邻域或基于领域的模型可以提高 *Tax* 数据集上的 F_1 得分。这是因为,在这个包含数千行的庞大数据集上,这些模型从数据错误的活跃域中提出了数千个可能修正的干净值。在这个巨大的搜索空间中,学习找到实际修正需要更多的学习迭代和用户标记。排除基于值的模型可以提高 *Hospital* 数据集的 F_1 评分。由于这个数据集随机添加了错别字,基于值的模型无法有效地从这种随机性中学习基于值的修正。这就是为什么排除基于值的模型时, *Hospital* 数据集的 F_1 评分更高。在列间依赖性较高的数据集(如 *Hospital* 和 *Flights* 数据集)中,排除基于邻域的错误校正器模型会显著降低 F_1 得分。这一下降表明,Baran 通过包含基于邻域的模型有效地修复了列间依赖性违规。在默认的错误校正器模型集中添加自定义数据集特定模型不会影响大多数数据集的 F_1 得分,因为我们的默认模型足够通用,已经涵盖了这些普遍的数据约束。例如,在基于邻域的模型中,属性对属性的功能依赖已经纳入我们的系统。

6.4 元组抽样影响分析

我们通过比较采用两种不同抽样方法的两种系统版本,分析了元组抽样方法对 Baran 算



(a) Hospital (b) Flights (c) Address (d) Rayyan

Figure 4: System effectiveness with different tuple sampling approaches.

法有效性的影响。均匀抽样方法根据均匀概率分布选择错误的元组进行用户标注。我们的元组抽样方法根据元组对分类器的信息量进行选择。

如图 4 所示,我们的元组抽样方法加快了系统的收敛速度。这种更高的收敛速度在数据错误随机分布的数据集(如 *Hospital* 和 *Rayyan*)上更为显著。在这些数据集上,分类器需要从每列和数据错误类型中获取足够多的标记数据错误,才能准确修复所有数据错误。这就是为什么我们的元组抽样方法对标记不足的列进行过度采样,从而更快地收敛。由于篇幅有限,我们仅报告了 4 个数据集的结果。在其余数据集上,Baran 的采样改进效果并不明显。

6.5 分类器影响分析

我们分析了分类器对 Baran 有效性的影响。我们测试了 *AdaBoost*、决策树、梯度提升和随机梯度下降,所有这些都在 *scikit-learn* Python 模块[33]中实现。我们应用网格搜索来为每个分类器找到最佳超参数。

Table 6: System effectiveness with different classifiers.

Classifier	Hospital			Flights			Address			Beers			Rayyan			IT			Tax		
	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F
AdaBoost	0.88	0.86	0.87	1.00	1.00	1.00	0.67	0.32	0.43	0.91	0.89	0.90	0.76	0.40	0.52	0.98	0.98	0.98	0.84	0.78	0.81
Decision Tree	0.88	0.85	0.86	1.00	1.00	1.00	0.70	0.34	0.46	0.91	0.89	0.90	0.62	0.34	0.44	0.98	0.98	0.98	0.74	0.73	0.73
Gradient Boosting	0.94	0.68	0.79	1.00	1.00	1.00	0.63	0.12	0.20	0.92	0.81	0.86	0.66	0.41	0.51	0.99	0.98	0.98	0.97	0.59	0.73
Stochastic Gradient Descent	0.95	0.92	0.93	1.00	1.00	1.00	0.66	0.25	0.36	0.95	0.87	0.91	0.59	0.21	0.31	0.99	0.98	0.98	0.83	0.63	0.72

表 6 显示,分类器的选择对系统的有效性没有显著影响。虽然在某些数据集上,如 *Address* 数据集, F_1 得分的变化更大,但总是有多个分类器能够达到几乎相同的 F_1 得分。在我们的当前原型中,我们部署了 AdaBoost,因为它是一个高级集成分类器[16],不易出现过度拟合[38]。

6.6 错误检测影响分析

尽管在文献中,错误检测和错误校正被认为是两个正交的任务[3, 20, 37, 48],但分析不完善的错误检测对下游错误校正有效性的影响非常重要。自然,错误校正的有效性取决于错误检测的有效性。通常,错误检测召回是错误校正召回的上限[37]。我们利用最先进的错误检测系统 Raha [27]来测试端到端数据清理管道的性能。Raha 也是一个免配置系统,只需要每个数据集的几个带标签的元组来检测数据错误。我们比较了三种端到端数据清理方案的有效性。第一种方案与之前一样,Baran 将完美检测到的数据错误作为输入并修复它们(表 7 中的行 *Perfect ED* + *Baran*)。在第二种方案中,Raha 检测数据集的数据错误,然后用户完美修复所有检测到的数据错误(行 *Raha* + *Perfect EC*)。这种虚拟

方法的有效性是纠错系统的上限。最后,在最后一个场景中,Raha 检测到数据错误,然后一个错误校正系统(如 Baran 或 HoloClean)修复检测到的数据错误。我们特别研究了 Raha 和 Baran 两个版本的端到端数据清理流水线的有效性。在第一个流水线中,Raha 和 Baran 是正交的,它们分别要求用户标记 20 个元组(行 Raha + Baran)。在第二个集成流水线中,只有 Raha 要求用户标记 20 个元组,然后将这些标签与检测到的数据错误一起传递给 Baran(行 *Raha + Baran(In)*)。我们还报告了 HoloClean 在处理同一组检测到的数据错误(行 *Raha + HoloClean*)时的有效性。

Table 7: System effectiveness with imperfect and perfect error detection (ED) and error correction (EC).

System	Hospital			Flights			Address			Beers			Rayyan			IT			Tax		
	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F
Perfect ED + Baran	0.88	0.86	0.87	1.00	1.00	1.00	0.67	0.32	0.43	0.91	0.89	0.90	0.76	0.40	0.52	0.98	0.98	0.98	0.84	0.78	0.81
Raha + Perfect EC	0.98	0.58	0.73	0.97	0.75	0.85	0.83	0.85	0.84	0.98	1.00	0.99	0.83	0.79	0.81	0.99	0.98	0.98	0.97	0.98	0.97
Raha + HoloClean	0.19	0.41	0.26	0.08	0.16	0.11	0.01	0.01	0.01	0.01	0.01	0.01	0.00	0.00	0.00	0.01	0.01	0.01	0.11	0.11	0.11
Raha + Baran	0.89	0.52	0.66	0.88	0.53	0.66	0.57	0.32	0.41	0.93	0.87	0.90	0.50	0.27	0.35	0.98	0.96	0.97	0.84	0.66	0.74
Raha + Baran (In)	0.95	0.52	0.67	0.84	0.56	0.67	0.53	0.32	0.40	0.93	0.87	0.90	0.44	0.21	0.28	0.99	0.97	0.98	0.84	0.77	0.80

如表 7 所示,不完善的错误检测自然会导致 Baran 的纠错效果略有下降。这种下降在大多数数据集(如 *Beers* 和 *IT*)上都是很小的。使用 Raha 和 Baran 的两种流水线几乎达到相同的 F_1 分数,并且都明显优于使用 HoloClean 的流水线。有趣的是,第二种流水线在大型数据集(如 *Tax*)上取得了更高的效果。这表明 Raha 基于聚类的采样[27]对于错误检测和校正任务的信息元组采样足够有效,并且我们不需要为 Baran 单独添加用户标签。

7. 相关研究

我们回顾了错误校正和迁移学习的相关研究,因为它们是这项工作的主要重点。此外,我们讨论了数据转换、示例编程、拼写检查和错误检测,因为我们的系统部分涉及这些领域。

错误校正。现有的错误校正系统利用各种信号,例如完整性规则[11, 13, 17, 18]、外部来源[12]、主动学习[24, 49]、清理样本[45]、统计可能性[48]以及规则和统计的组合[37]。这些方法在数据冗余和用户提供的规则和参数可用的情况下很有前景。Baran 提供了一种新的任务公式,不需要这些先决条件。

迁移学习。迁移学习已被用于实体匹配[50]、缺失值插补[44]和性能估计[26]。Baran 是第一个利用迁移学习进行错误校正任务的系统。

数据转换和编程举例。数据转换是将数据值从一种格式转换为另一种格式的任务[6, 23, 5]。编程举例是合成满足一组输入输出示例的程序的任务[19, 39, 40]。Baran 利用数据转换和编程举例作为错误校正的一种类型,即基于值的校正。

拼写检查。拼写检查是指识别并修正文本中的错别字(即拼写错误)[34, 28]。Baran 不仅通过基于值的模型修正拼写和其他语言错误,还修正其他类型的数据错误,例如缺失值和格式问题[36]。

错误检测。错误检测是指检测错误数据值的任务[3]。以往的方法利用各种技术来检测数据错误,例如数据增强[20]、网络表格[46]、元数据[41]、主动学习[29]以及组合错误检测算法[27]。Baran 的任务与错误检测任务正交,因为任何错误检测方法的输出都可以作为输入提供给 Baran。

8. 结论

我们提出了一种新的纠错系统,该系统根据数据错误的值、邻域和领域上下文来修复数据错误。Baran 根据这些不同的上下文训练多个纠错器模型,然后将它们组合成一个最终纠错器,用于纠正每个数据错误。此外,Baran 还提供了迁移学习的选项。正如我们的实验所显示的那样,Baran 的性能明显优于现有的纠错系统。尽管 Baran 前景广阔,但仍有待改进。特别是设计一个有效的数据清理仪表板,这是一个重要的方向,可以帮助用户避免更正错误。