实验：

在这一部分，我们选取了来自不同数据源的，数据量差异较大的多种数据进行了实验分析。我们与kraska在2017年提出的RMI模型，在查询时间，内存占用，查询准确率方面，进行了对比实验。

## Dataset：

我们采用了来自真实场景下的多种数据，如表一所示。表中列出了数据集的名字，数据集大小，存储的数据类型，以及数据中总的属性数量。book\_uint32是来自某购物网站的2018年图书销售情况的收集，它根据售出图书所属的学科或种类，比如生物类，历史类，漫画类等，作为表中数据的属性，划分一年内所有售出的图书，并且记录下每个书售卖的数量，共计200M条记录。该表中数据类型为unsigned int类型，表中数据关联较少，因为较少出现某一种类的书热销，使得另一种类的书也随之热销的情况。数据中的噪音也相对较少。Book\_float64是基于上述图书的销售数量，加入了书籍单价的属性，最终存储的value为书的售卖总金额。这里书籍出售总数和书籍销售总金额，有着较高的共线性，可以用来验证上文提到的利用正则项改进后模型的效果。Wiki中存储的是2012年间wiki百科关于特定话题的一些网页信息，共计350M条，在表中用String类型进行存储。我们还采用了经典的Iris 鸢尾花数据集。数据集内包含 3 类共 150 条记录，每类各 50 个数据，每条记录都有 4 项特征。从而，我们覆盖了整形，浮点型，字符串等基本的查询数据类型。此外，我们还使用一些已知数据分布的数据集，包括随机分布，正态分布，高斯分布，指数分布等等，基本覆盖了常见的数据分布类型，保证模型在各个场景下的应用效果。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| name | type | items | size | from |
| Iris | Unsigned int 64 | 150 | 5KB | sklearn.datasets.load\_iris |
| Book\_uint32 | Unsigned int 32 | 200 million | 762M | http://www.dangdang.com/ |
| Book\_ufloat64 | Unsigned float 64 | 200 million | 762M | http://www.dangdang.com/ |
| wiki\_ts | String | 200 million | 1.48G | Wikipedia |
| osm\_cellids\_uint64 | Unsigned int 64 | 500 million | 1.92G | OpenStreetMap |
| exponential\_s | Unsigned int 64 | 500 million | 1.94G | Self-produce |
| normal\_s | Unsigned int 64 | 160 million | 608M | Self-produce |

表一

## 实验环境介绍：

本实验为对比试验，实验中我们提出的架构的测试数据来自centos8.0.1905系统平台下，Intel Core i7-9750 with 3.60 GHz CPU, 16GB memory，配置gcc，python3.8以及一些其他依赖包后的测试。对于kraska提出的RMI模型，我们在centos中建立了rust环境，利用cargo命令，对于相同的数据集，处理成二进制文件后进行测试。

## 训练过程：

数据集本身为csv格式，需要经过处理与读取，首先进入学习索引模块，利用递归的层次结构，逐层缩小待查询的数据所在的相关字段的范围。模型每层不保存数据，只需要向下传递参数，包括预测后的相关字段的上界，下界，以及这样估计后的误差的上下界。这一过程内存占用较多，主要体现在数据处理与顶层的一些大型模型的训练，可达300M至600M；底层的模型自动选择模块，在训练过程中，由于需要训练多个回归模型以及随机森林分类器，内存占用较高。对于kraska提出的RMI模型，提供的输入文件类型为：带有指定后缀的二进制文件，可以采取普通的人工选取模型，进行叠加的层次训练，或者是采取cargo命令行参数，让系统自行选取合适的层，包括样条插值函数，三阶样条插值，线性回归，逻辑回归等，加入到模型中进行优化。采用在cargo命令行中加入参数的方式，我们可以得到kraska模型给出的最优化方案的json文件，经过对比json文件与我们提出的模型在训练时间，采取的模型，内存占用的差异，我们针对训练过程有如下发现：

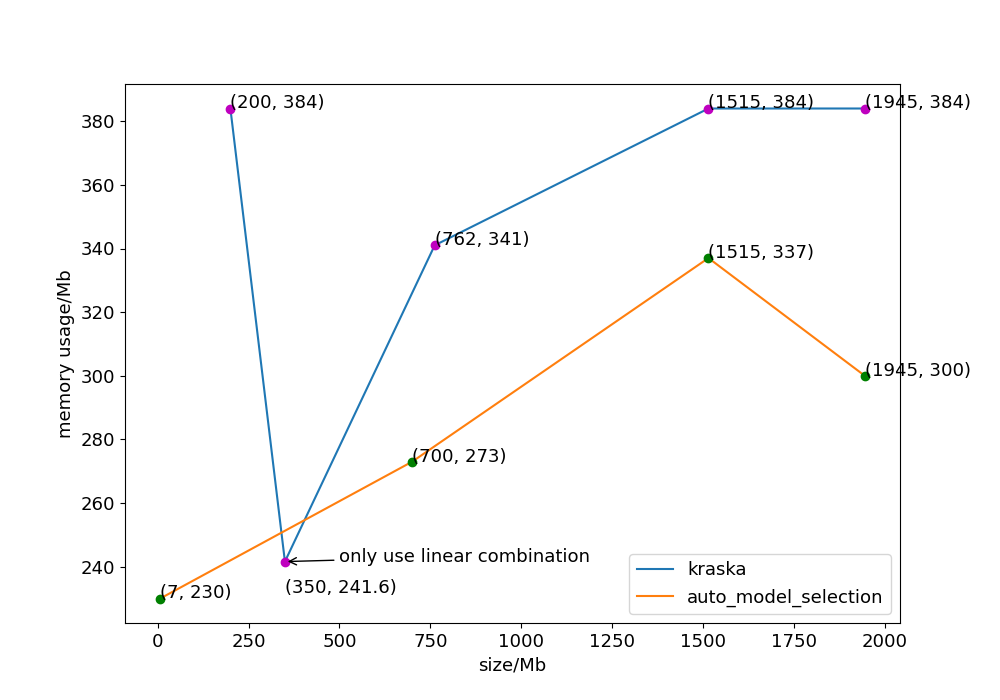
**实验发现：**

1. 输入文件采取二进制文件的方式，可以显著减少由于数据处理消耗的时间与内存，但是这一操作反过来成为提升查询准确率的瓶颈（因为在特殊情形下，需要回表时这样的数据进行查询十分困难，我们在后面会提到）。我们需要在保证准确率的前提下，尽可能的减少内存占用。2. 对于我们的模型，训练时间基本只与数据的大小以及数据类型相关，模型表现较为稳定，人为干预因素较少。而对于kraska提出的模型，不仅与数据量及数据类型密切相关，而且严重依赖与人为选择的模型的组合性能的好坏。人为选取模型进行组合，使得训练时间大大快于kraska中根据策略选取最优模型组合的方式，但是训练得到的效果也更差。这说明在此条件下，已有的kraska模型的训练效果，严重依赖于人的专业知识，可用性较差。

## 查询过程实验：

**实验一：内存占用对比：**

首先，针对不同大小的数据集，我们对模型的内存占用情况进行对比试验。Rmi中可供选择的模型都是单个内存占用低，相对简单的弱学习器回归模型或者多阶样条函数。它采用多层叠加的方式提高总体效果，并且提供了两种模式：人为自主添加与自动优化机制。为了说明模型效果，我们采用加入了自动优化进行训练的RMI模型进行对比。在我们的auto\_model\_selection模型中，我们采用占用单个内存占用相对较高，但是拟合效果很好的多种回归模型。实验效果如下图所示：

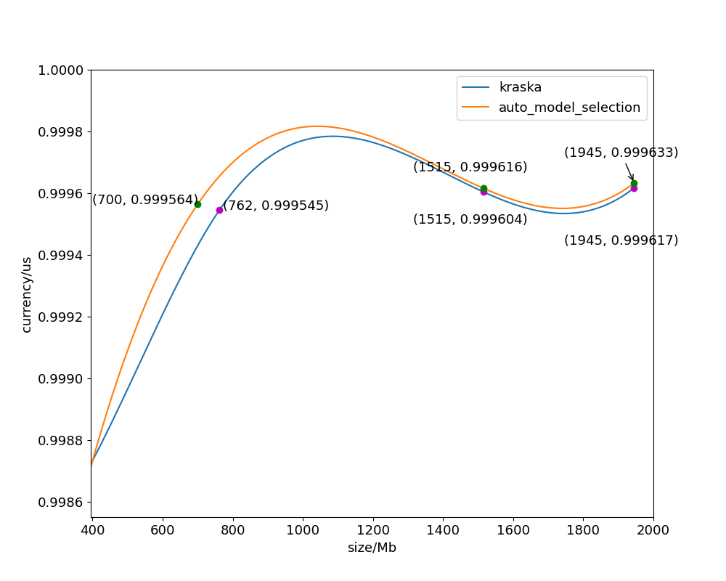


图中，横轴是选择的数据集的大小，单位为Mb，纵轴是内存占用情况，单位为Mb。橙色线条代表我们的自动选择模型，蓝色的是kraska的rmi模型。图中的点是我们记录的实验结果在实验中，对于采用的随机分布数据（200，384），wiki（1515，384），正态分布数据（1945，384），三者采用优化等级最高的rmi模型与我们训练得到的最优模型进行了对比试验。实验表明，rmi优化等级最高情况下，内存占用基本固定为384M，为4层rmi（4-stage rmi）。此时由于rmi的层数较多，大量模型的堆积使得其内存占用情况超过了我们的自动选择模型挑选出的单个回归模型的内存占用。因此我们的模型在此方面表现得更加优秀。

另一方面，为了权衡模型其他效果（准确率和查询时间）与模型内存占用哪一个更需要优化，我们设计：将我们挑选出的模型与rmi中只用内存占用最低的线性回归模型叠加而成的组合层次结构进行对比试验。观察到点（350，241.6）处，rmi的内存占用略低于我们的自动选择模型。然而在接下来的实验中，我们将会观察到，这点处的查询时间rmi的时间约等于我们模型的二倍！（见实验三）。此外，为了满足查询准确率的要求，RMI对于查询错误进行修复时，需要付出额外的内存占用，（详见实验二），使得它在数据处理方面的优势被削弱。综上所述，我们在内存上的优化的优越性得到了保证。

**实验二：查询准确率对比**

下一步，我们对于查询结果准确率进行了对比试验，实验结果如下图所示：

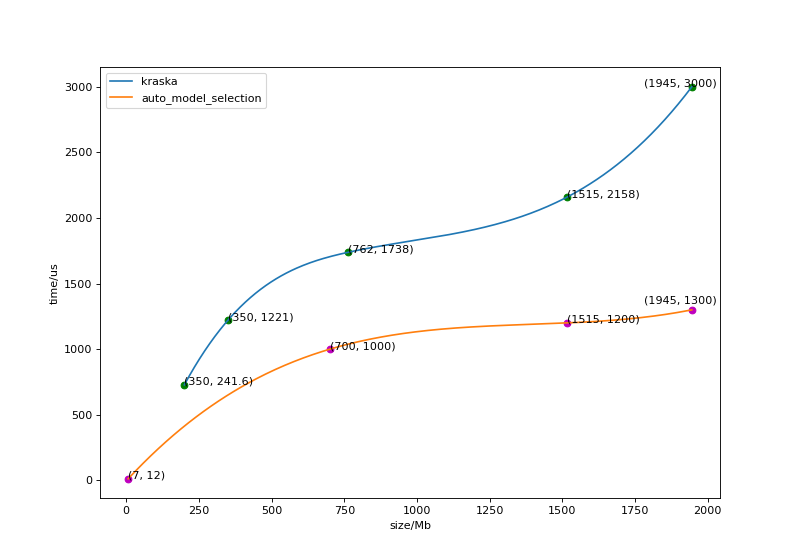


可以观测到在400M到1200M之间，我们的模型准确度比kraska模型都有着较大程度的提高，尤其是在600M到1000M之间，能高出约为0.0001。在1200M之后，自动选择模型的准确率略微高于rmi模型。这是由于大量数据集训练下，我们采用的优秀的回归模型比如梯度提升树（GBDT）与极限树（ExtraTree）等，能有更丰富的训练数据集进行学习，从而获得更好的训练效果。同时，数据集增大后，自动选择模型中的数据经过重分布后，附加在key上的模型标签更加多种并且可靠，这些数据输入到分类器后，分类器训练效果得到了提高，因此在查询时，总体的准确度得到了一定程度的提高。总体来看，我们的模型的准确率在全程都优于rmi。

另一方面，从如何不足“最后一英里”的角度看，两个模型都需要对于余下的0.0004左右的误差进行处理修复。然而，现有的修复技术或者需要在架构的底层，也就是b数的叶节点中二分遍历查找，或者需要回表进行核对。然而，对于现有的RMI模型，数据输入时二进制形式，首先需要进行数据的还原，然后进行配对，查找等工作。这样的工作，虽然从整体查询来看几率较低，但是对于大规模的数据而言，这确实增加了额外的内存占用，使得RMI在输入数据处理方面的优势被削弱。

**实验三：查询时长对比：**

最后，我们对二者的查用用时进行了对比实验。由于rmi模型经过4层的学习，查询的结果是逐层输出的。而我们的自动模型选择模型，查询结果不用经过所有回归模型的训练，直接到分类器中寻找最优匹配的回归模型然后进行训练，并且模型的性能更高，因此在理论上，我们的查询时间更短。为了验证这个理论，我们对比试验的结果如下图所示：



正如实验一最后讲述的，在（350，241.6）处，对于350M的数据集，rmi最终查询时间约为我们模型的二倍。根据上图可以看出，rmi的模型在1500M之前约为对数增长趋势，在1500M之后斜率增大，有近似于指数的增长趋势；对于我们的模型，保持着略慢于对数增长的趋势，这是由于我们某些回归模型，其本质是基于bagging的树形结构。观察到，在1500M之后，点（1515，2158），（1945，3000），这两处的查询时间约为同等大小数据集下，我们模型用时的两到三倍，因此，面对容量较大的数据集，我们的模型在查询时间上有着显著的优势。