## 基于机器学习的数据库基数估计任务实验报告

**1 实验任务背景介绍**

基于机器学习的数据库基数估计任务试验背景介绍 在数据库领域，基数估计对查询优化至关重要，其旨在预估查询结果集的元组数。 传统基数估计方法依靠统计信息与启发式规则运作。在数据量较小、结构简单的时期，这类方法尚可满足基本需求。它们采集数据的基础统计数据，例如列值分布、不同值数量等，再凭借既定数学模型与规则推算查询涉及的数据量。但随着时代发展，数据状况发生巨大变化。如今的数据不仅规模庞大，还呈现出复杂特性，像多峰分布、强相关性、高度倾斜以及动态变化等情况屡见不鲜。传统方法基于简单统计与固定规则构建的估算模型，难以精准捕捉这些复杂的数据内在模式，致使基数估计误差较大。不准确的估计传递给查询优化器后，会造成执行计划制定失误，如连接顺序、索引策略或数据访问路径选择不当，最终导致查询效率低下，耗费过多系统资源，拖慢数据库整体性能与响应速度，难以应对当下大规模数据快速处理与实时分析的需求。 机器学习的兴起为数据库基数估计带来转机。机器学习算法擅长挖掘数据隐藏的复杂关系与模式，无需人工设定复杂规则与固定模型。它借助大量历史查询数据、样本数据及元数据展开学习训练，构建贴合数据特性的基数估计模型。像深度学习中的神经网络，可通过多层结构深度抽象与提取特征，捕捉传统方法无法察觉的模式与关联，使基数估计更精准。基于机器学习的方法有望提升数据库在复杂数据场景下的查询优化水平，增强系统资源利用率与性能，推动数据库技术在大数据时代的进一步发展与应用。

**2 相关技术介绍**

一、数据预处理与特征工程技术

数据采样：

由于数据库数据量通常非常庞大，在训练机器学习模型时可能无法直接使用全部数据。数据采样技术可以从原始数据中抽取具有代表性的子集。例如，随机采样可以简单地按照一定比例随机选取数据；分层采样则根据数据的某些特征（如数据类型、表的分区等）进行分层后再采样，确保每个层次都有足够的样本用于模型训练，这样可以在不损失太多信息的情况下减少训练数据量，提高训练效率。

特征编码：

数据库中的数据类型多样，包括数值型、字符型、日期型等。需要将这些不同类型的数据转换为机器学习模型能够处理的数值特征。例如，对于字符型数据，可以使用独热编码（One-Hot Encoding）将其转换为二进制向量，每个可能的字符值对应一个向量位置；对于数值型数据，可以进行归一化或标准化处理，使其取值范围在特定区间内，避免不同特征取值范围差异过大对模型训练的影响。

特征选择与降维：

数据库查询相关的特征可能非常多，并非所有特征都对基数估计有重要作用。特征选择技术可以筛选出与基数关系密切的特征，如基于信息增益、卡方检验等统计指标来评估特征的重要性并选择重要特征。降维技术如主成分分析（PCA）则通过线性变换将高维数据映射到低维空间，在保留数据主要信息的同时减少特征数量，降低模型的计算复杂度并提高模型的泛化能力。

二、模型评估与优化技术

评估指标：

常用的评估指标包括均方误差（MSE）、平均绝对误差（MAE）等。MSE 衡量预测值与真实值之间误差的平方和的平均值，对较大误差更为敏感；MAE 则是预测值与真实值误差的绝对值的平均值，更直观地反映平均误差大小。通过这些指标可以评估机器学习模型在数据库基数估计任务中的准确性，以便选择合适的模型和调整模型参数。

超参数调整：

机器学习模型中有许多超参数，如神经网络中的层数、每层神经元数量、学习率等。超参数调整技术可以通过网格搜索、随机搜索或基于贝叶斯优化的方法来寻找最优超参数组合。例如，网格搜索会遍历所有指定的超参数取值组合，在训练集上训练模型并在验证集上评估，选择性能最佳的组合；贝叶斯优化则根据已有的评估结果建立概率模型，更智能地选择下一组超参数进行尝试，提高超参数调整的效率。

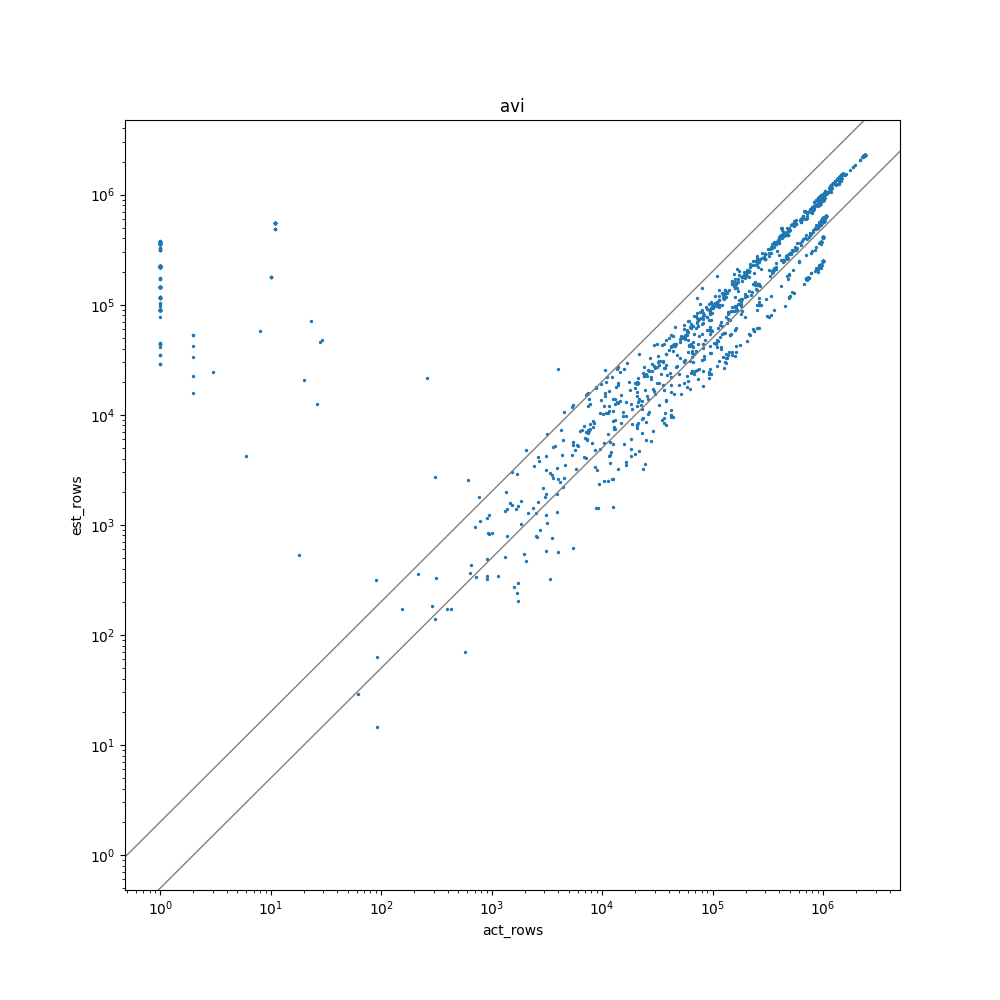
模型集成：

为了提高基数估计的准确性和稳定性，可以采用模型集成技术。如 Bagging 方法通过对训练数据进行多次有放回采样，训练多个基模型（如决策树），然后综合这些基模型的预测结果（如取平均值或投票）；Boosting 方法则是依次训练多个弱模型，后一个模型重点关注前一个模型的错误预测样本，逐步提升整体模型性能。模型集成可以融合多个模型的优势，减少单个模型的误差和过拟合风险。

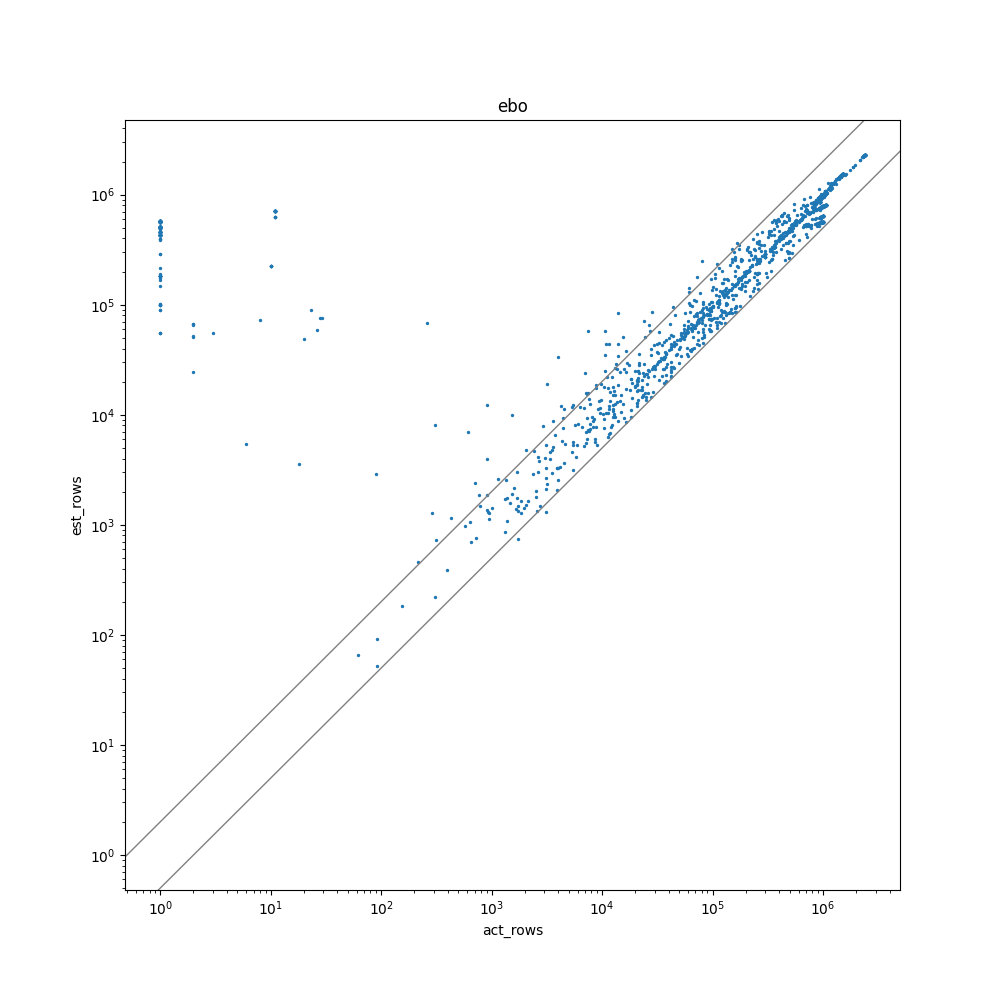
**3机器学习建模与实验测试：**

1. **线性回归模型：**

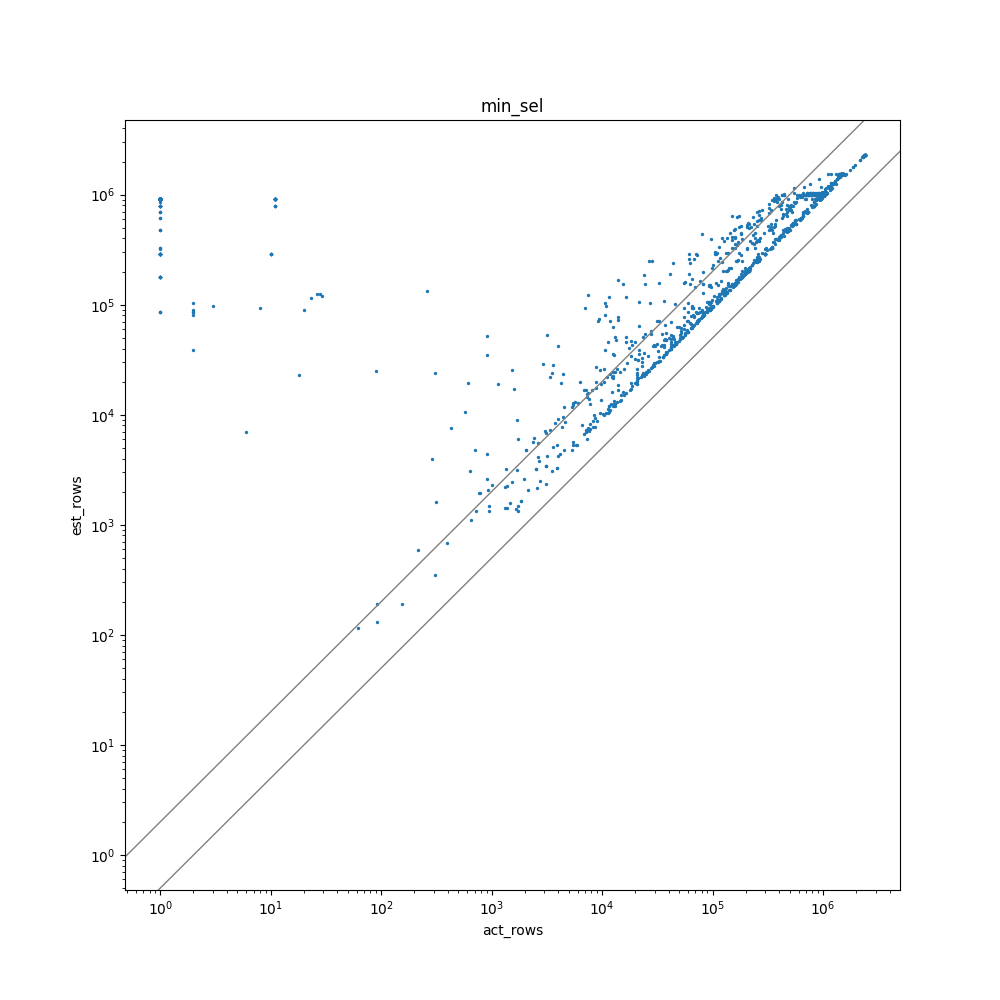
（A）AVI方法：



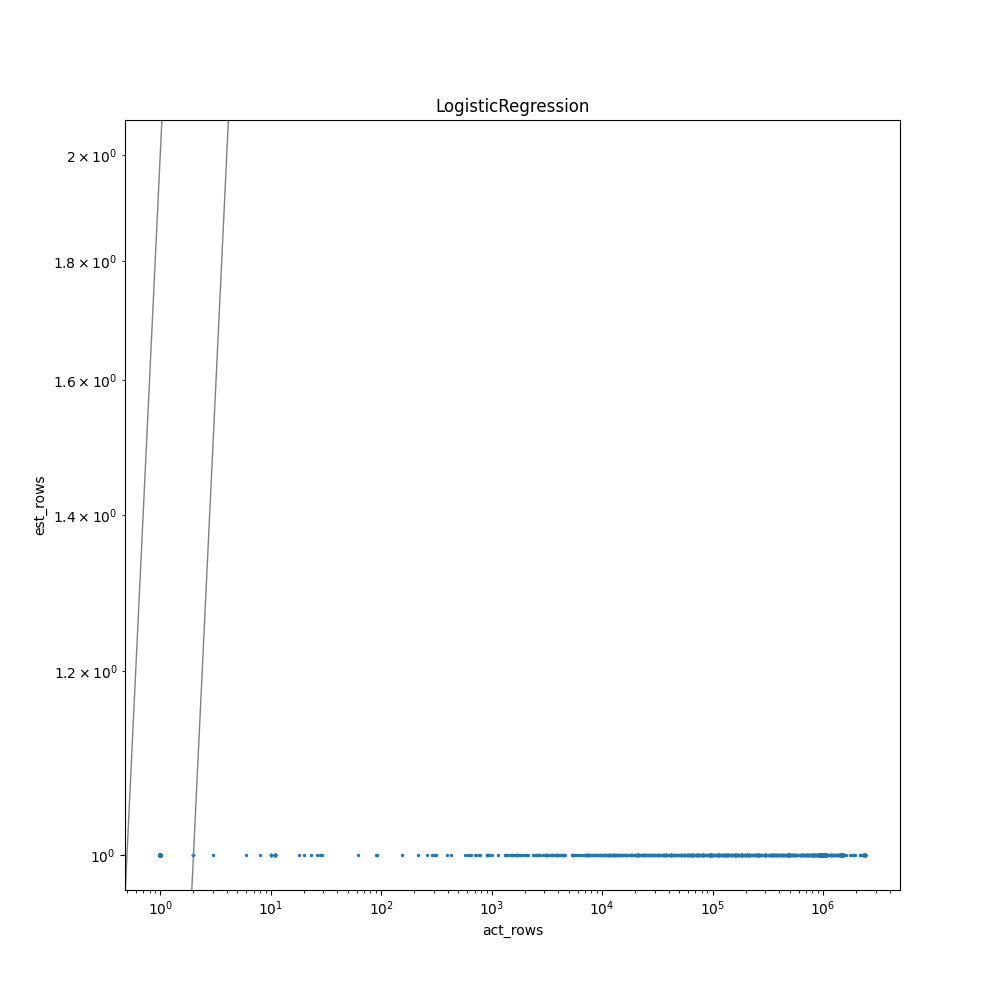
（B）EBO方法：



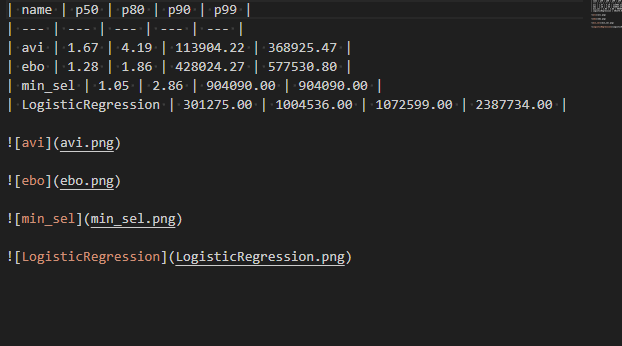
1. MinSel方法：



（D）Logistic Regression：

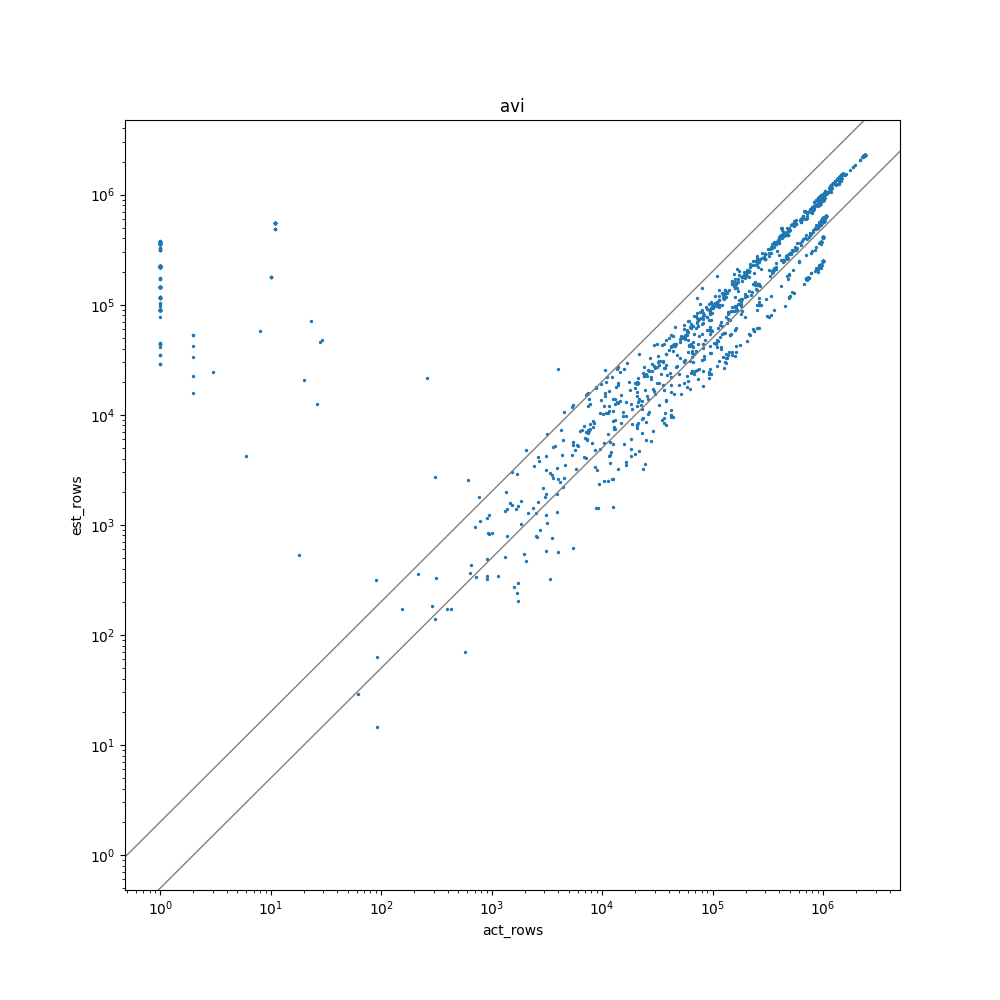


（E）代码运行截屏：

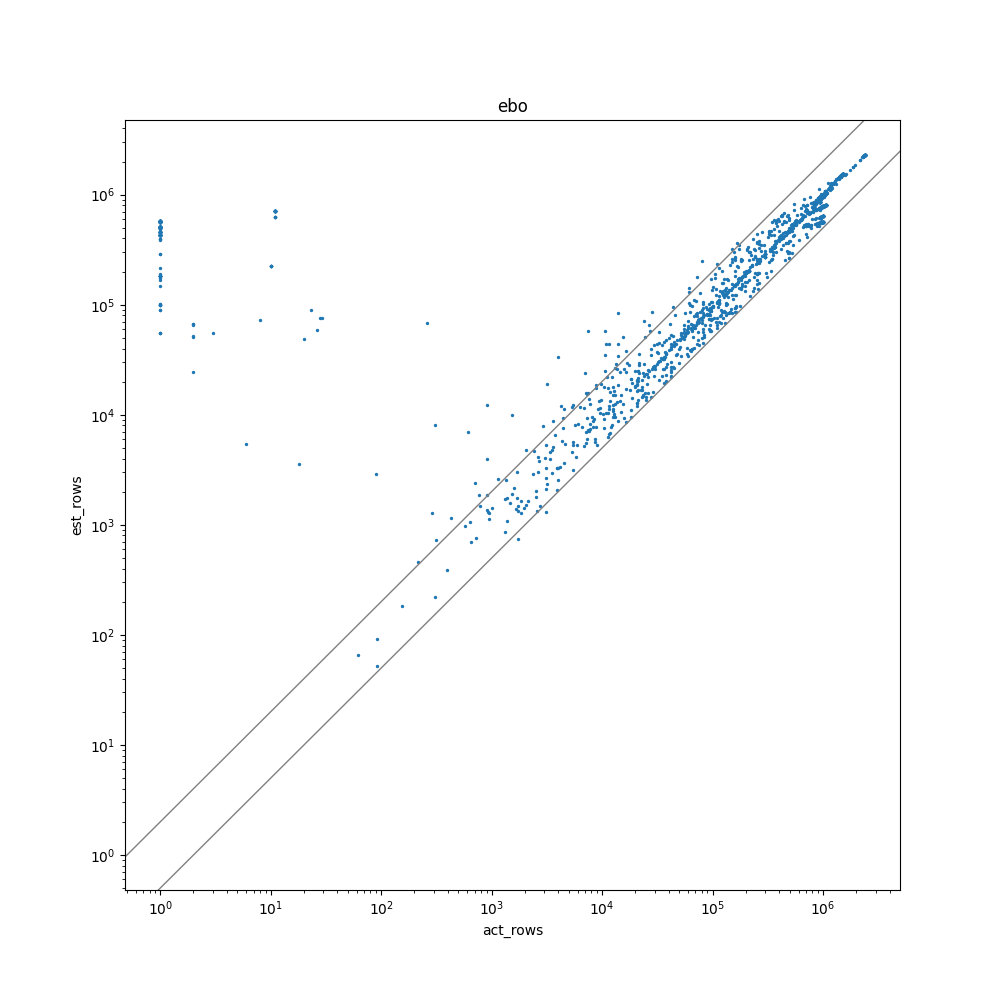


1. **SVM模型：**

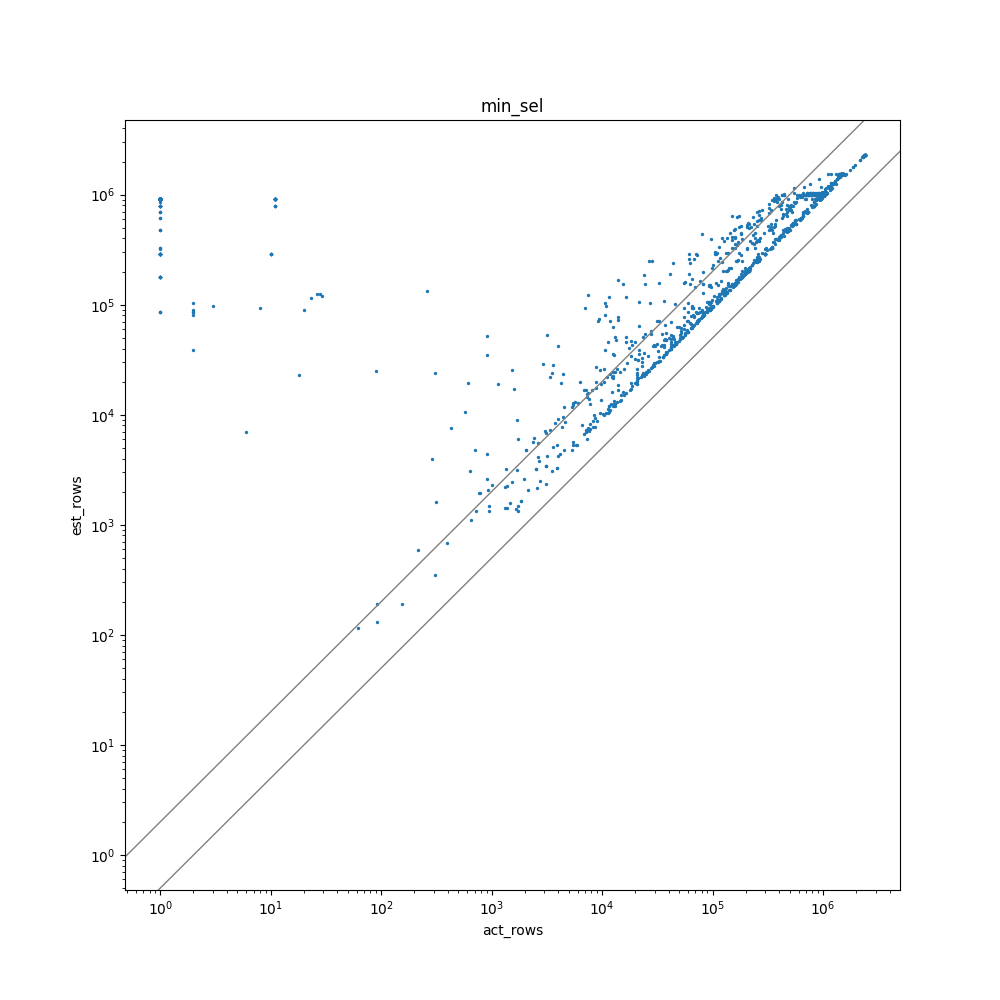
（A）AVI方式：



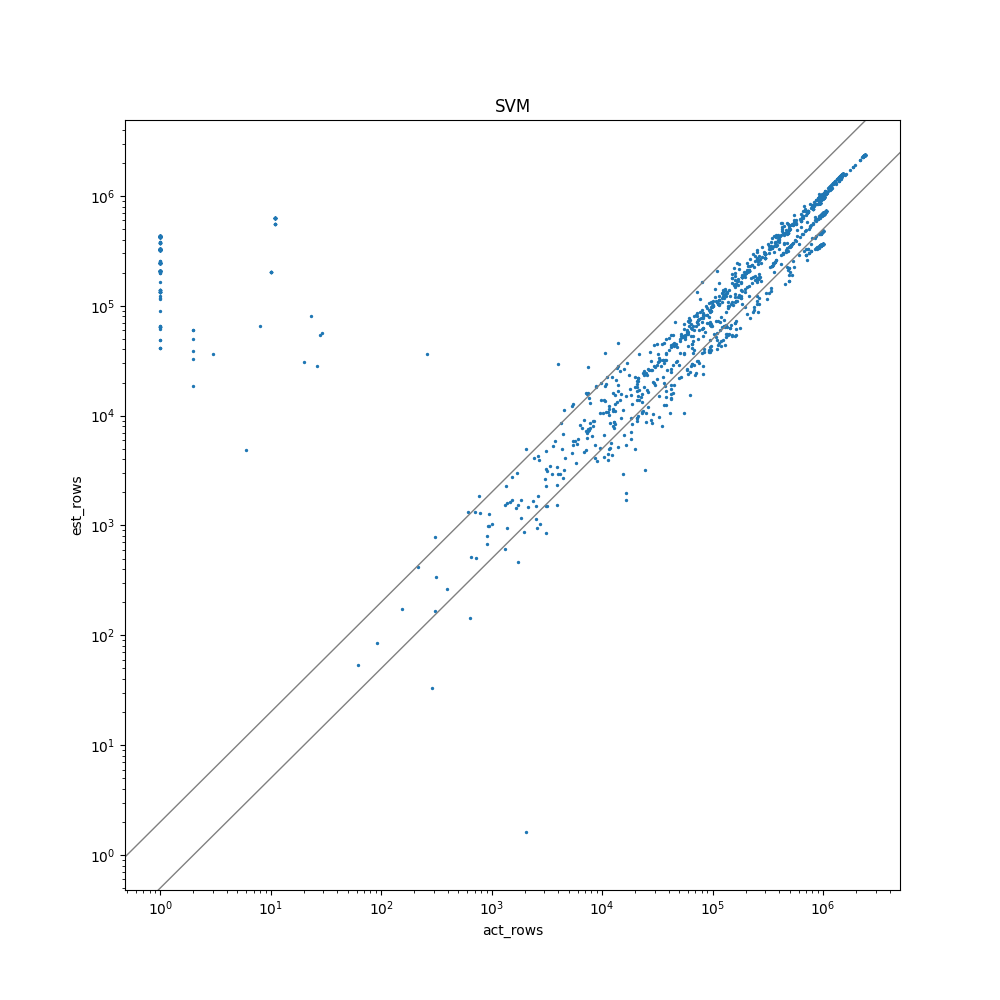
1. EBO方式：



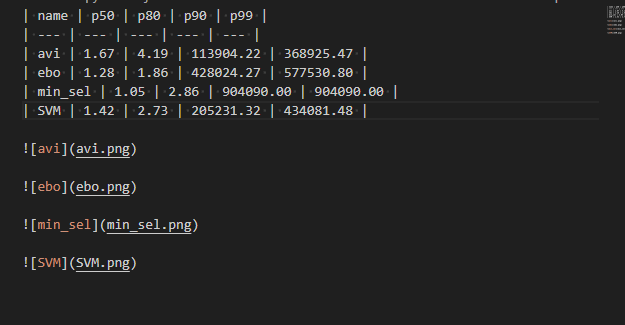
1. MinSel方式：



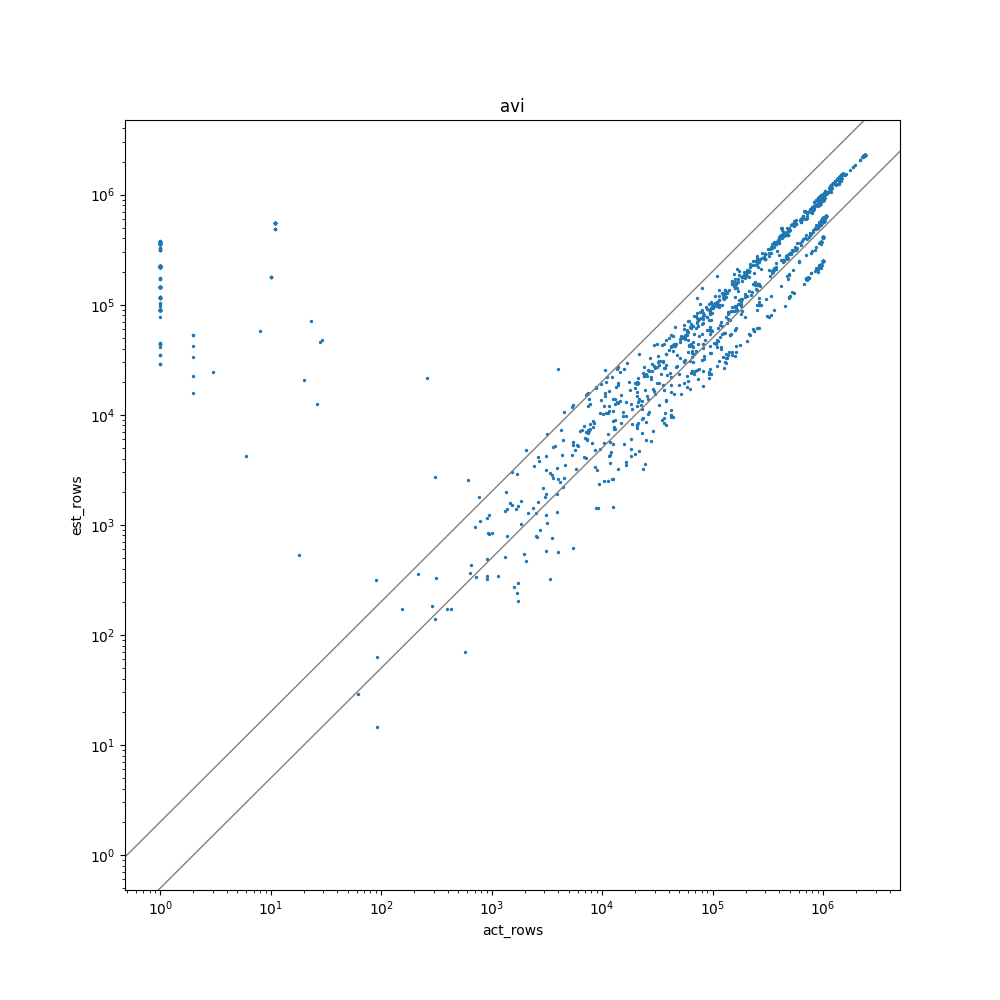
1. SVM：



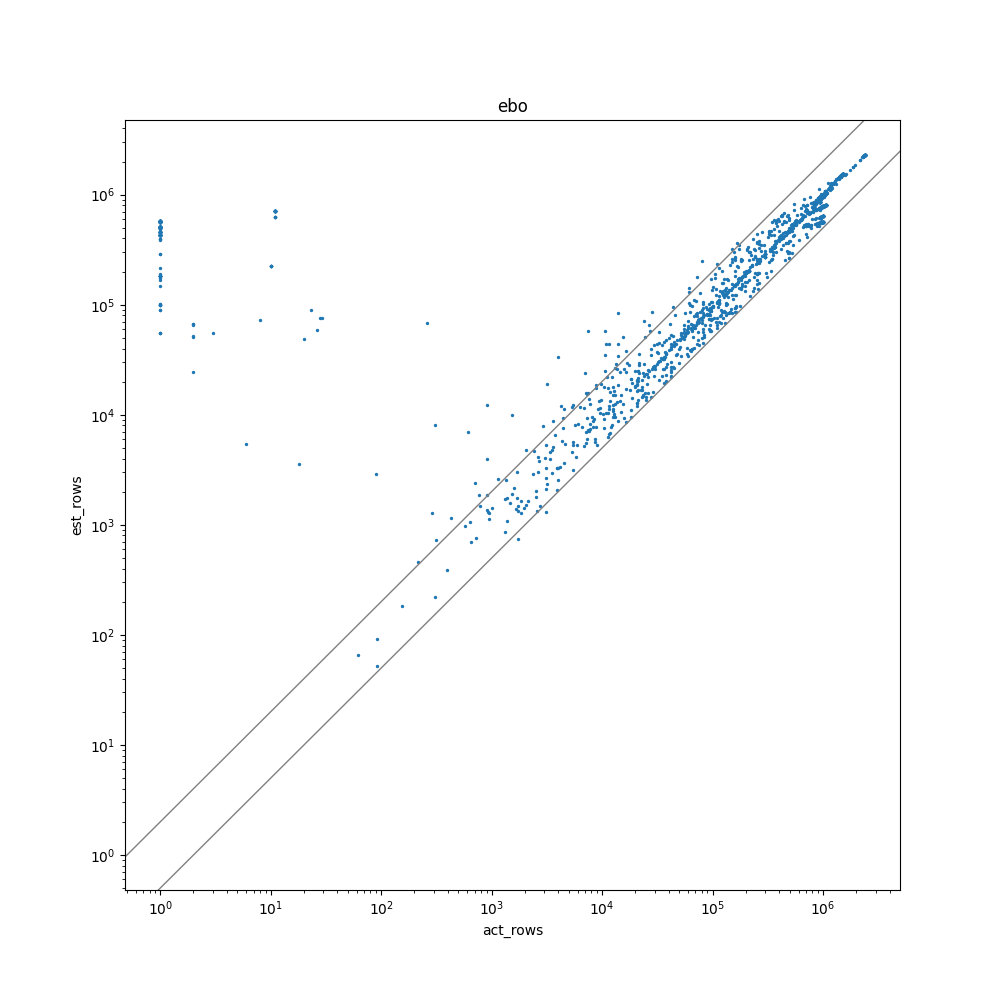
1. 运行截屏：



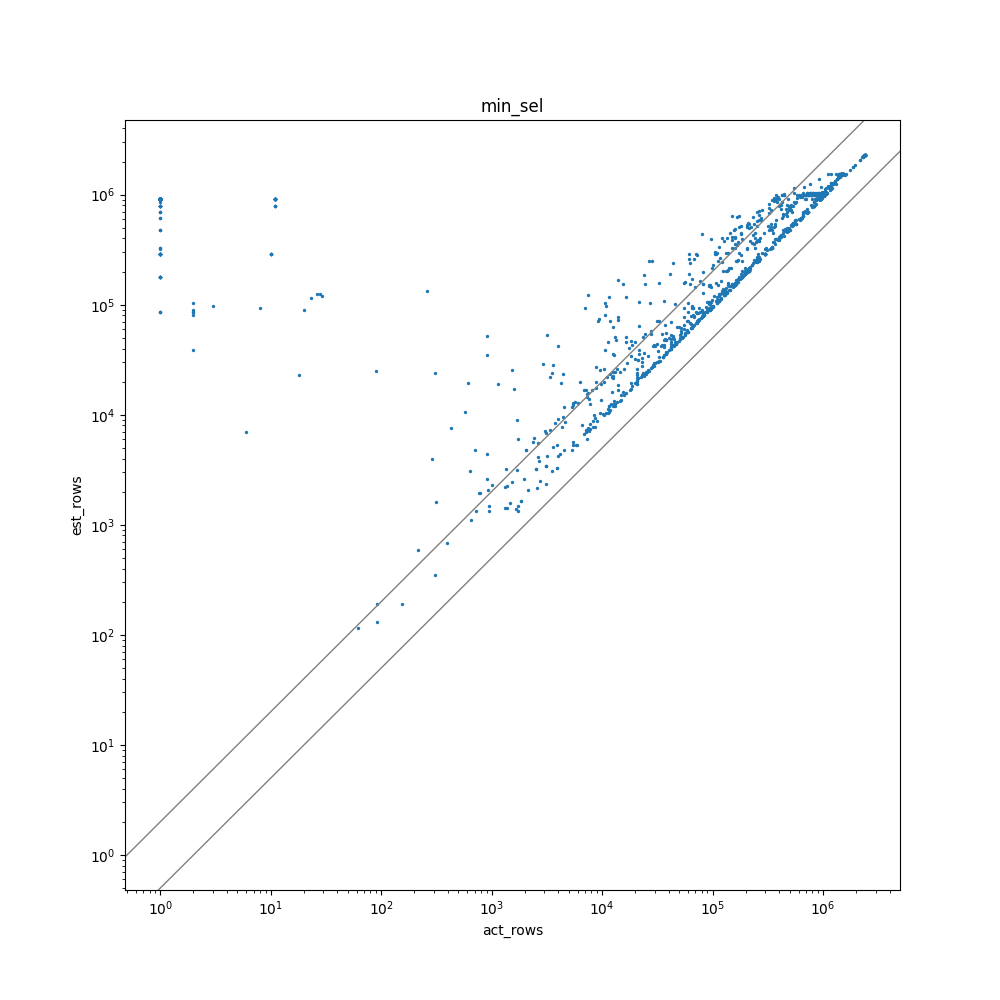
1. **随机森林**
2. AVI方式：



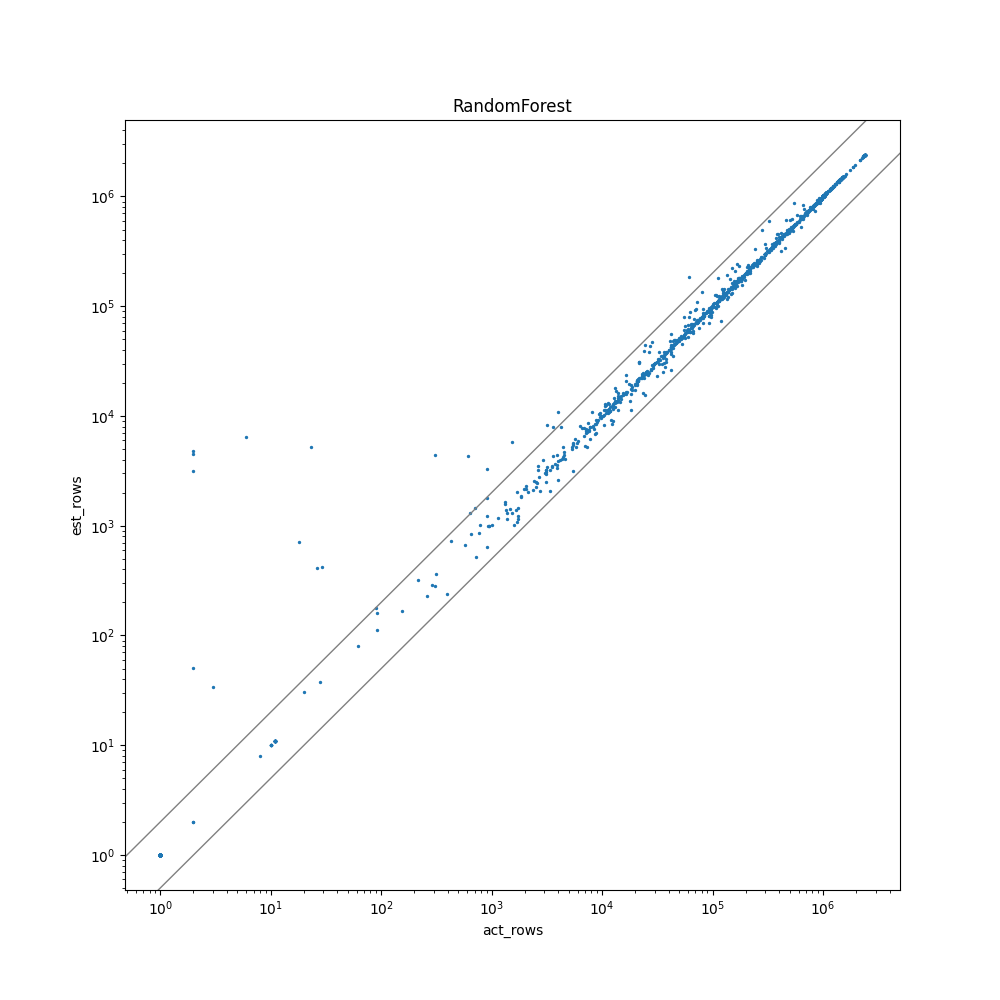
1. EBO方式：



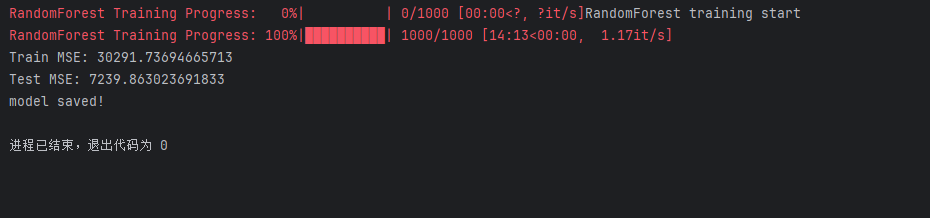
1. MinSel方式：

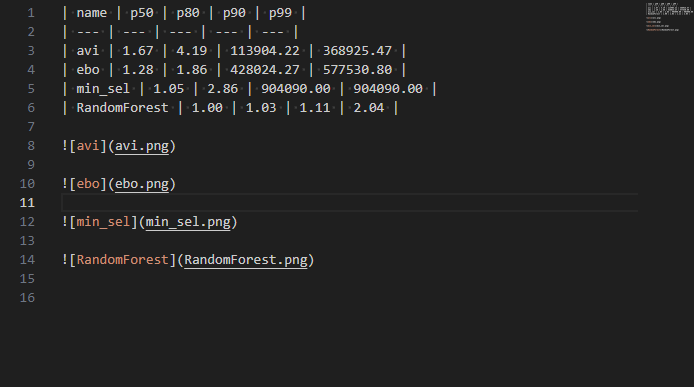


1. RandomForst：

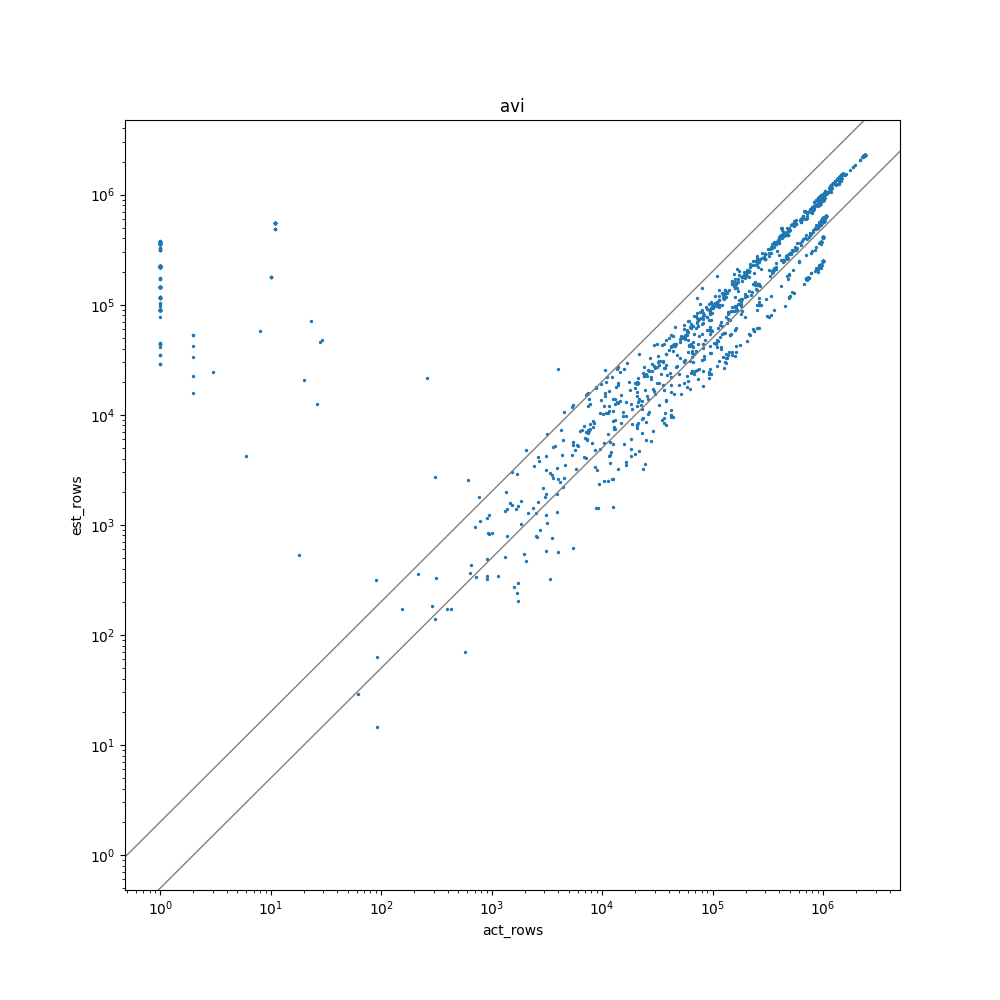


1. 运行截屏：

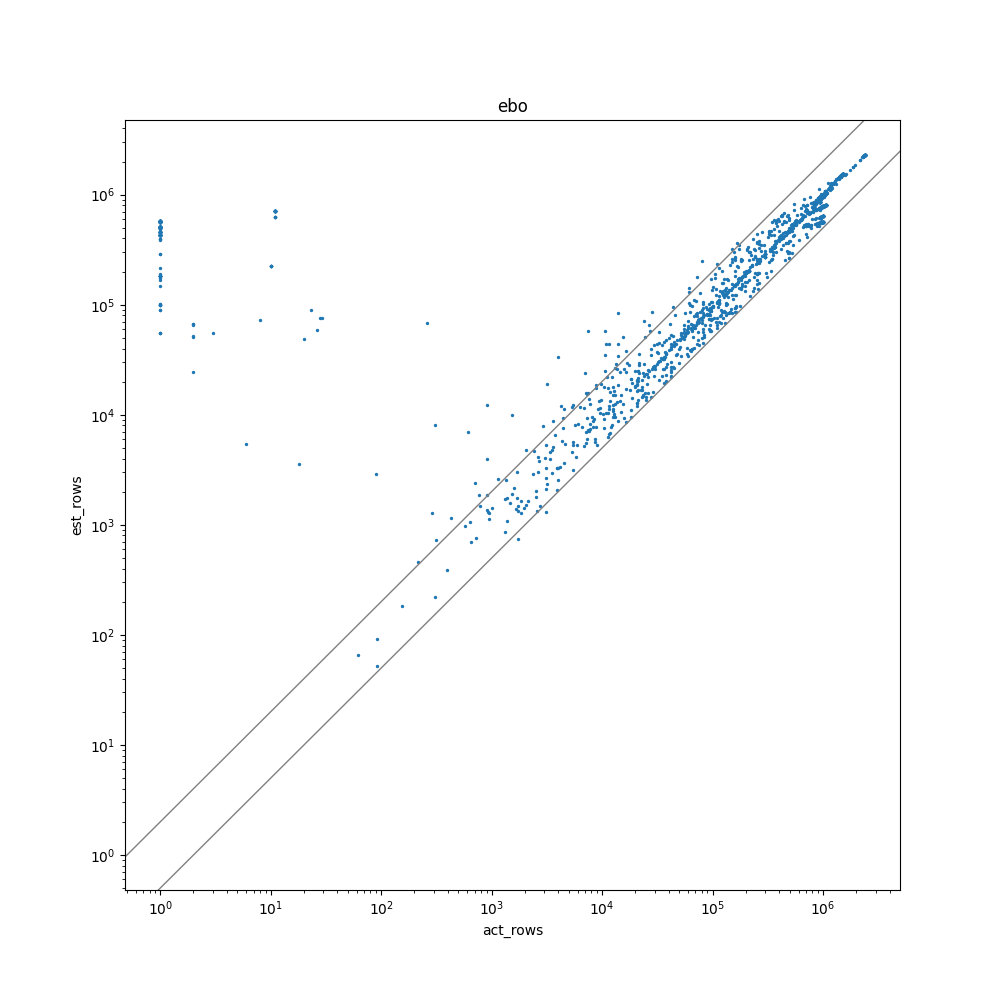




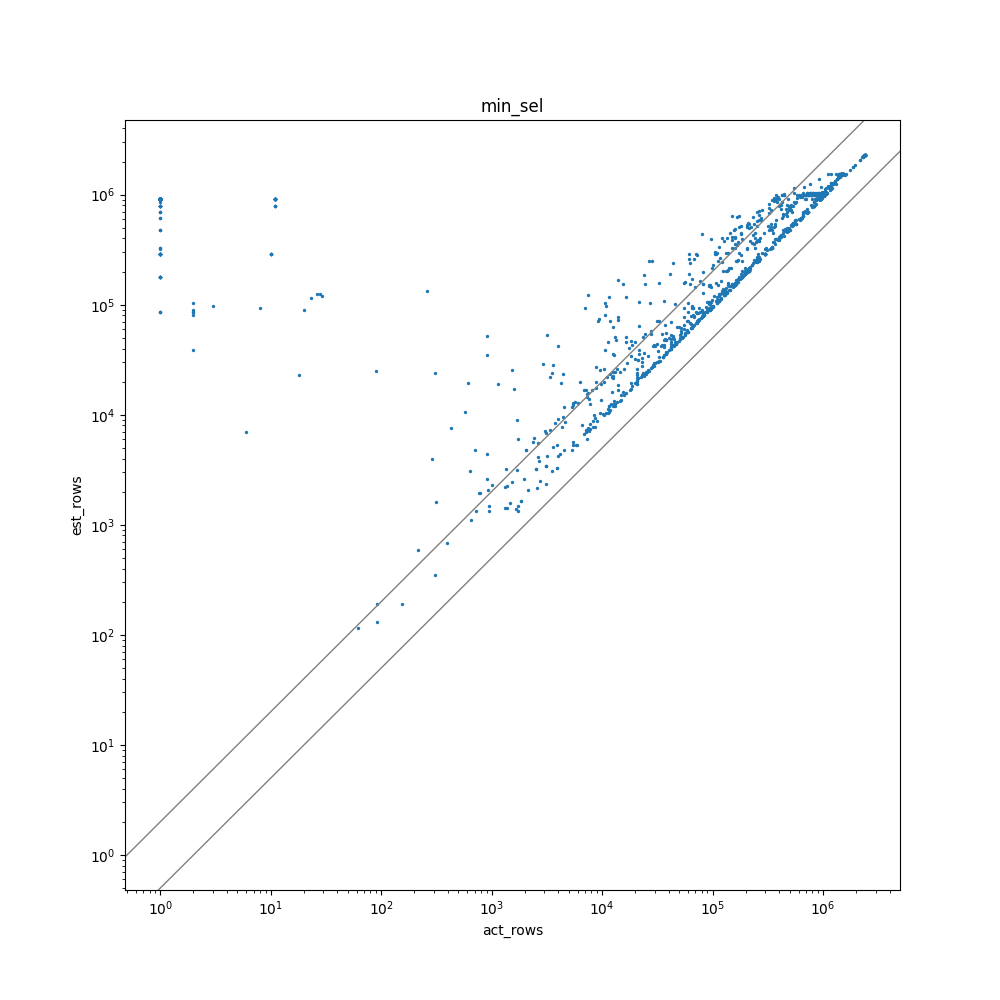
1. **XGBoost：**
2. AVI方式：



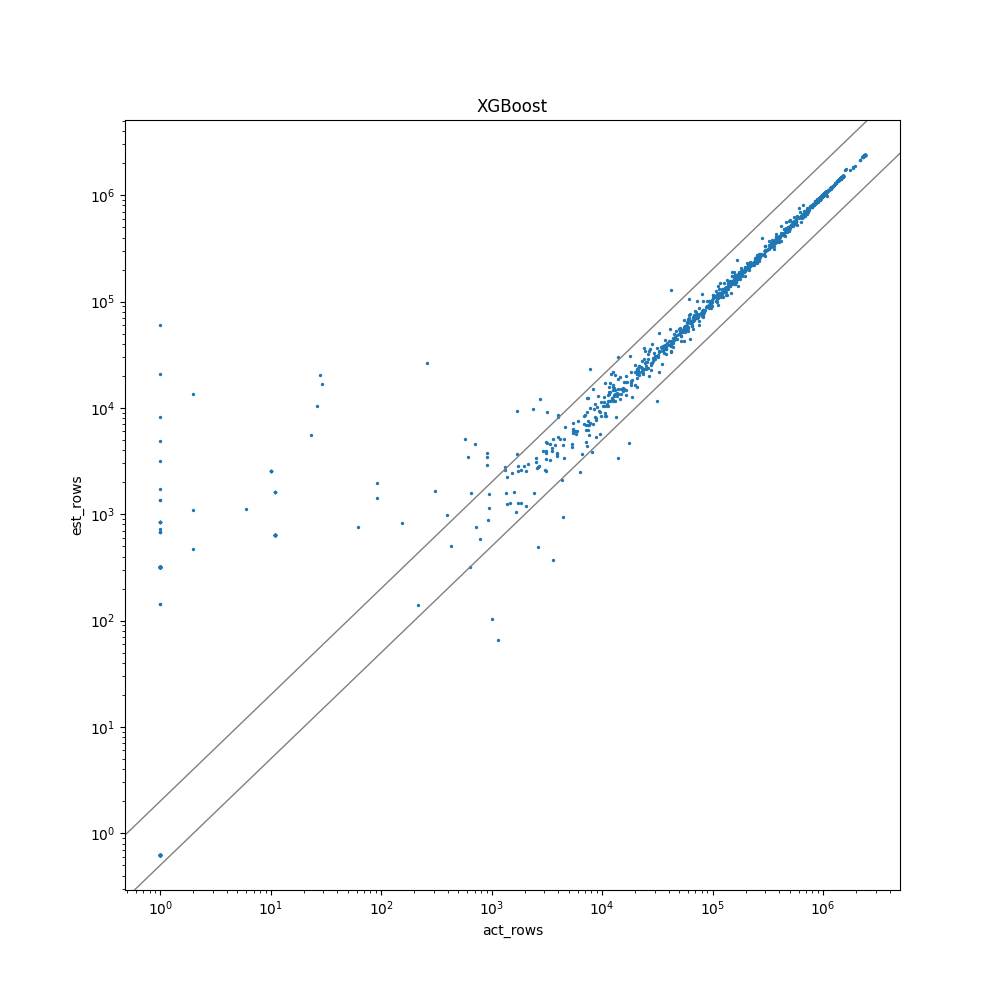
1. EBO方式：



1. MinSel方式：

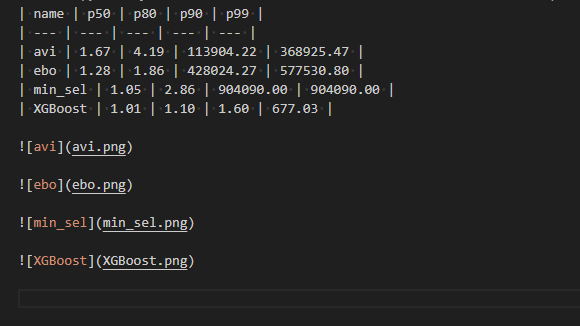


1. XGBoost：

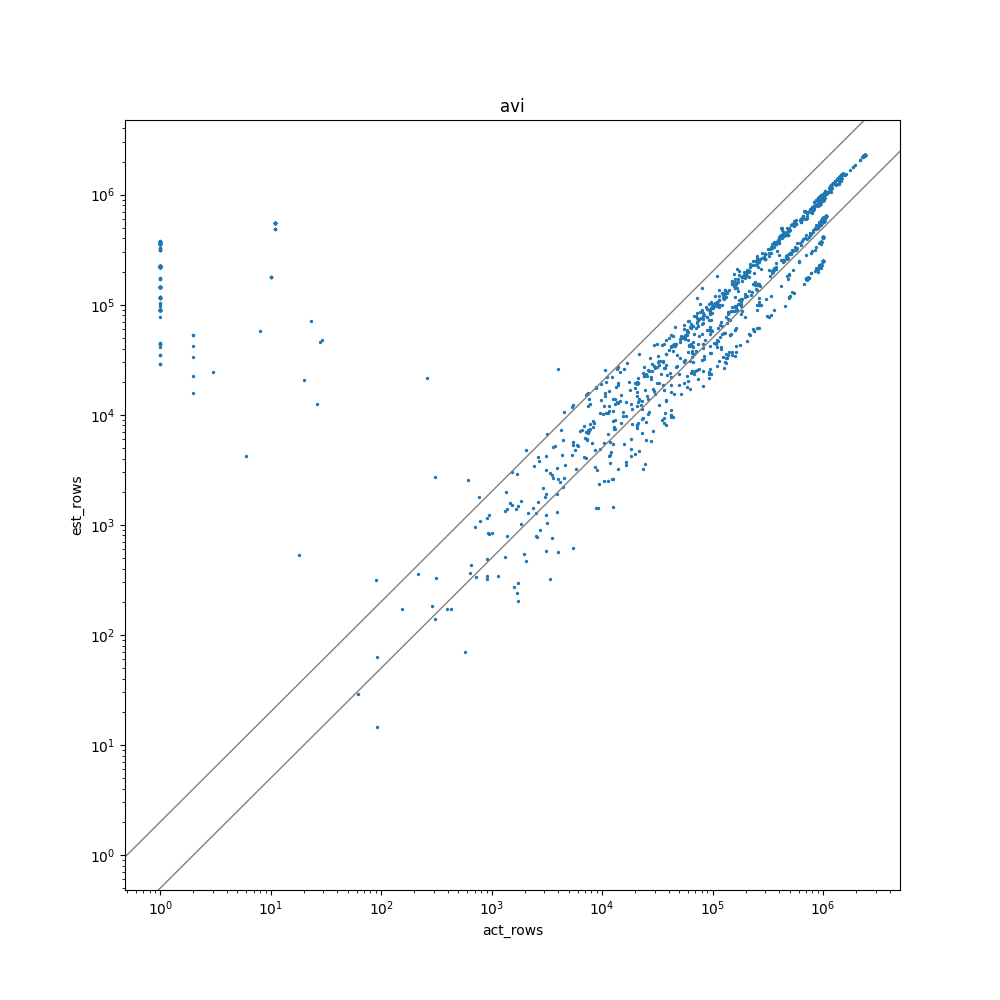


1. 运行截屏：

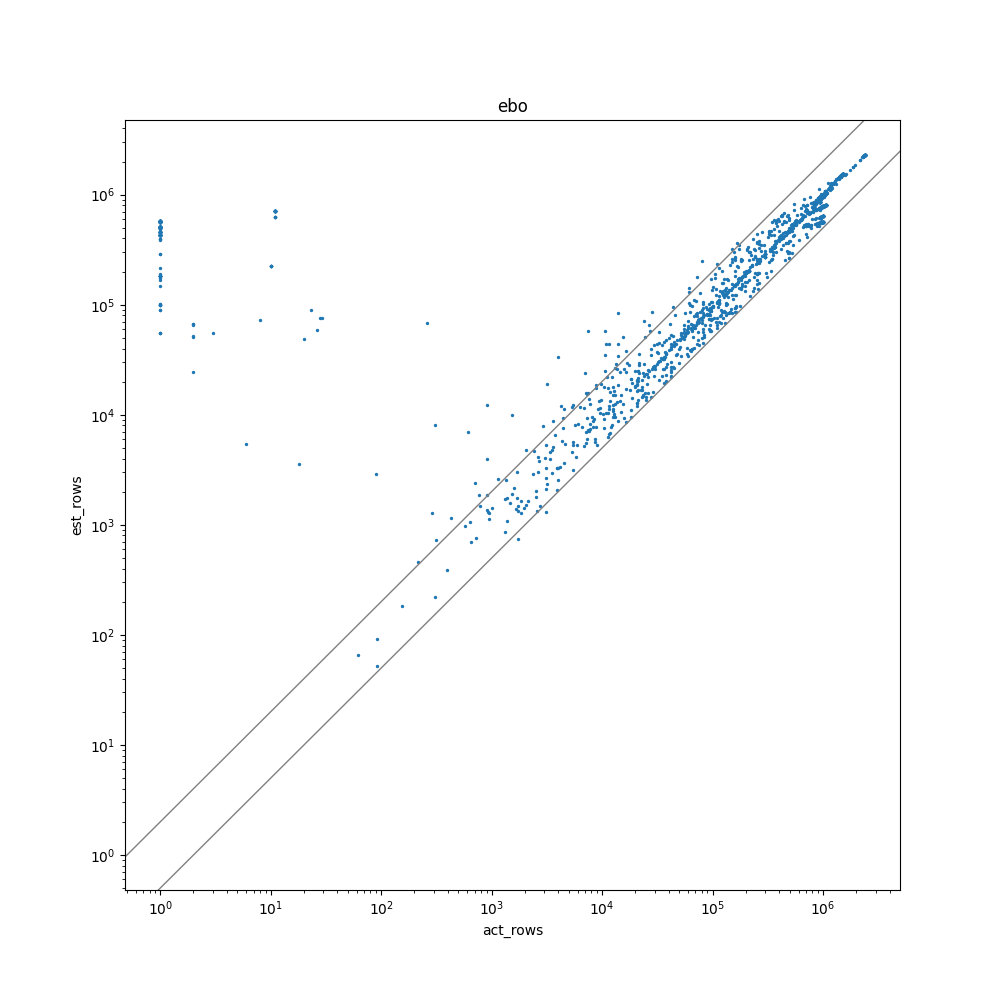




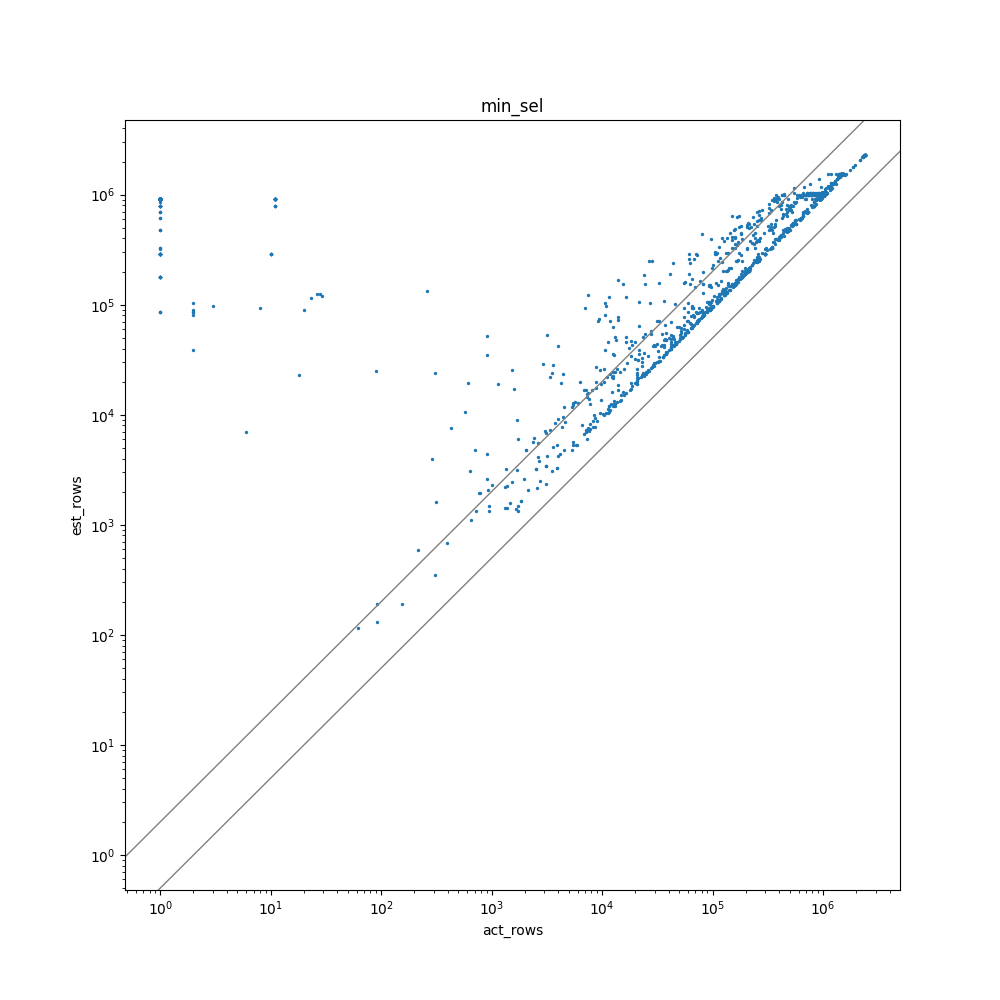
1. **MLP模型：**
2. AVI方式：



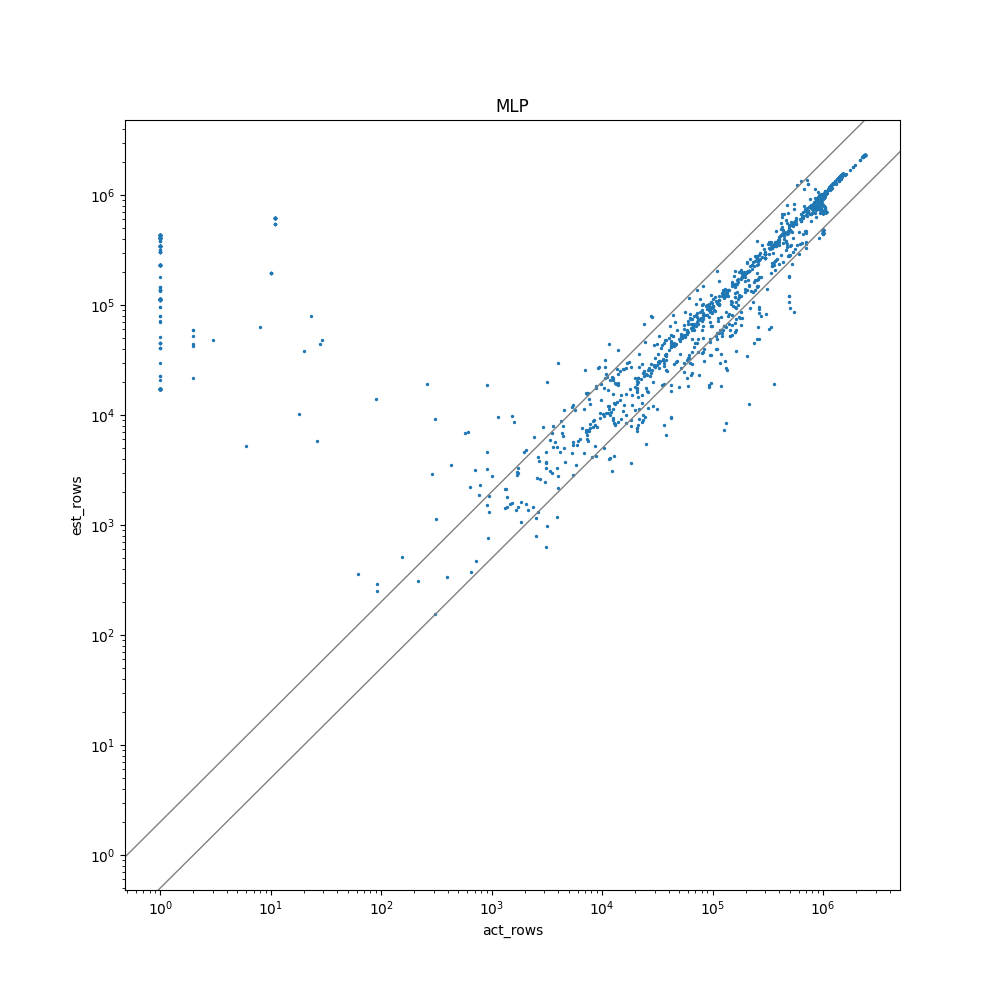
1. EBO方式：



1. MinSel方式：

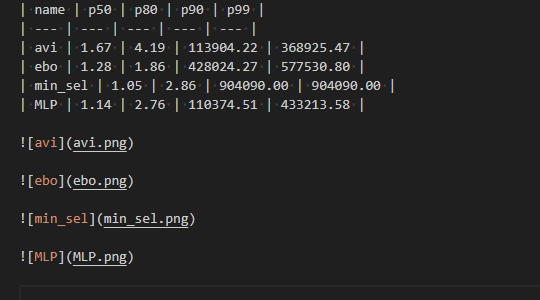


1. MLP模型：

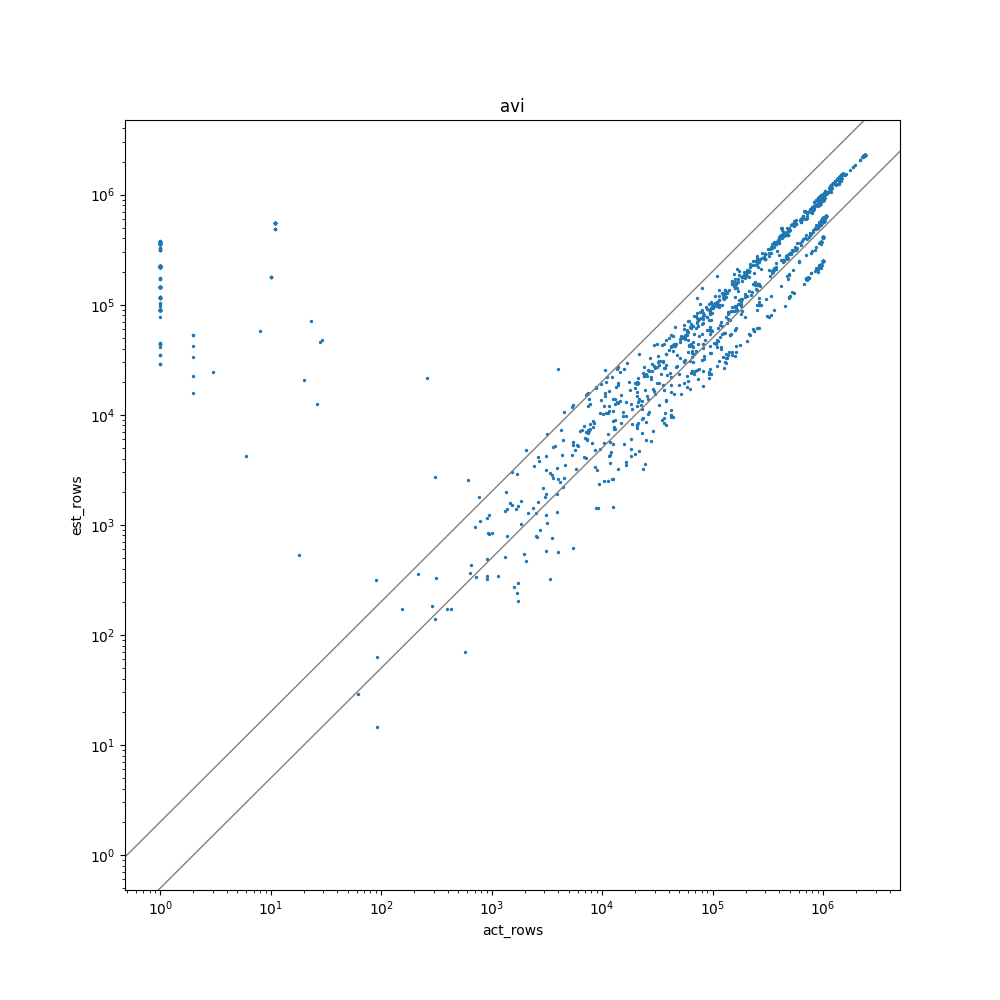


1. 运行截屏：

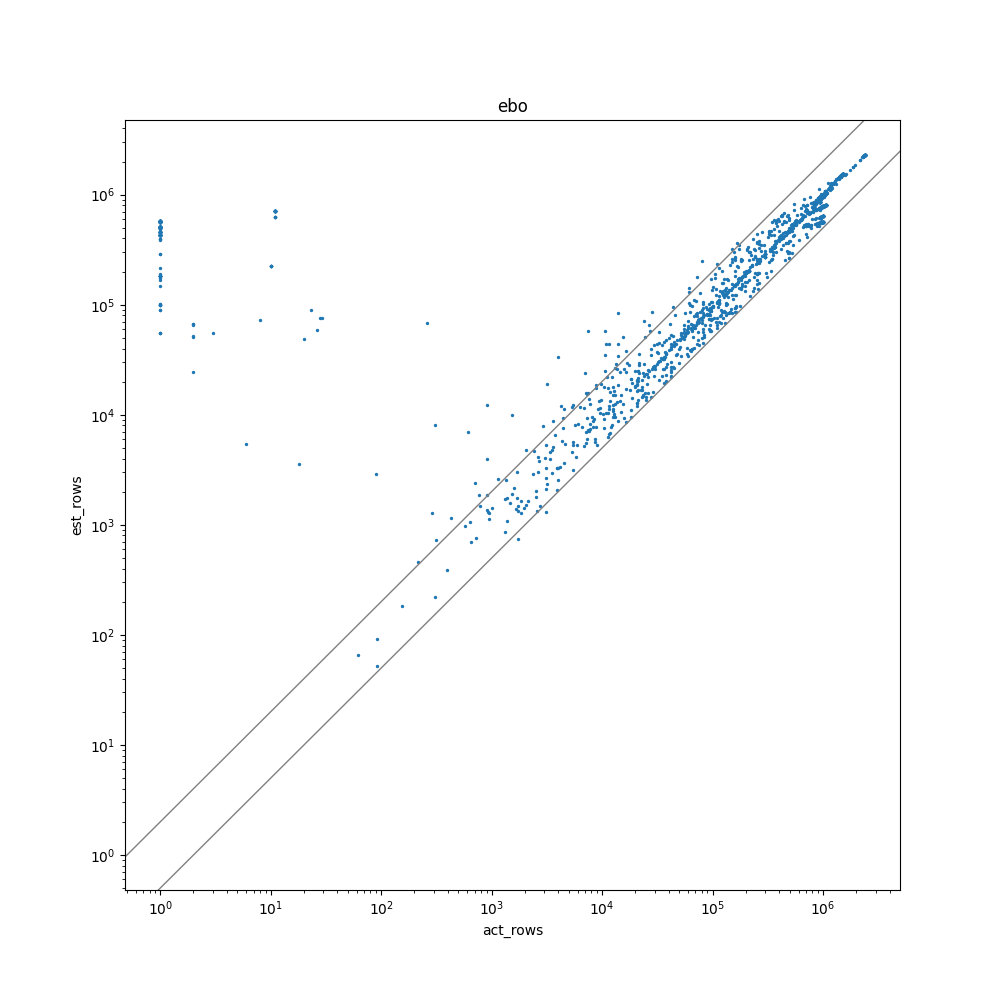




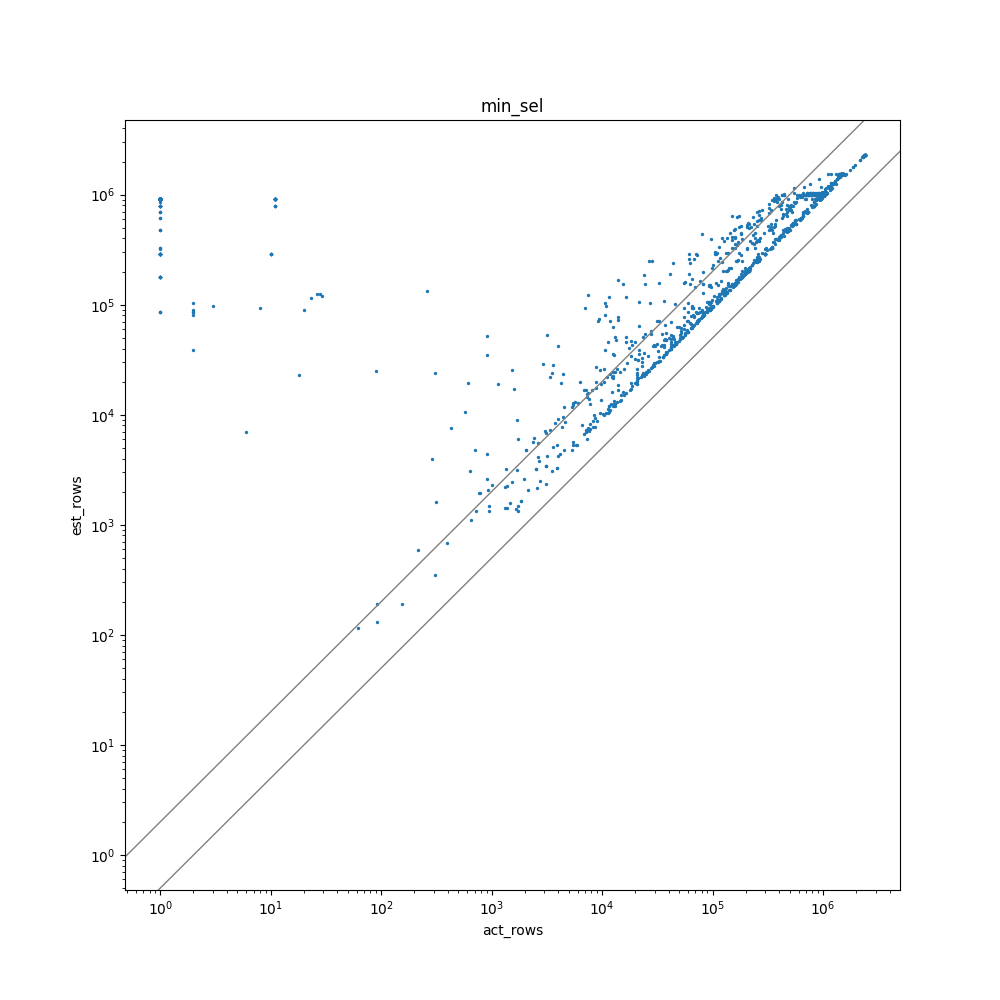
1. KNN模型：
2. AVI方式：



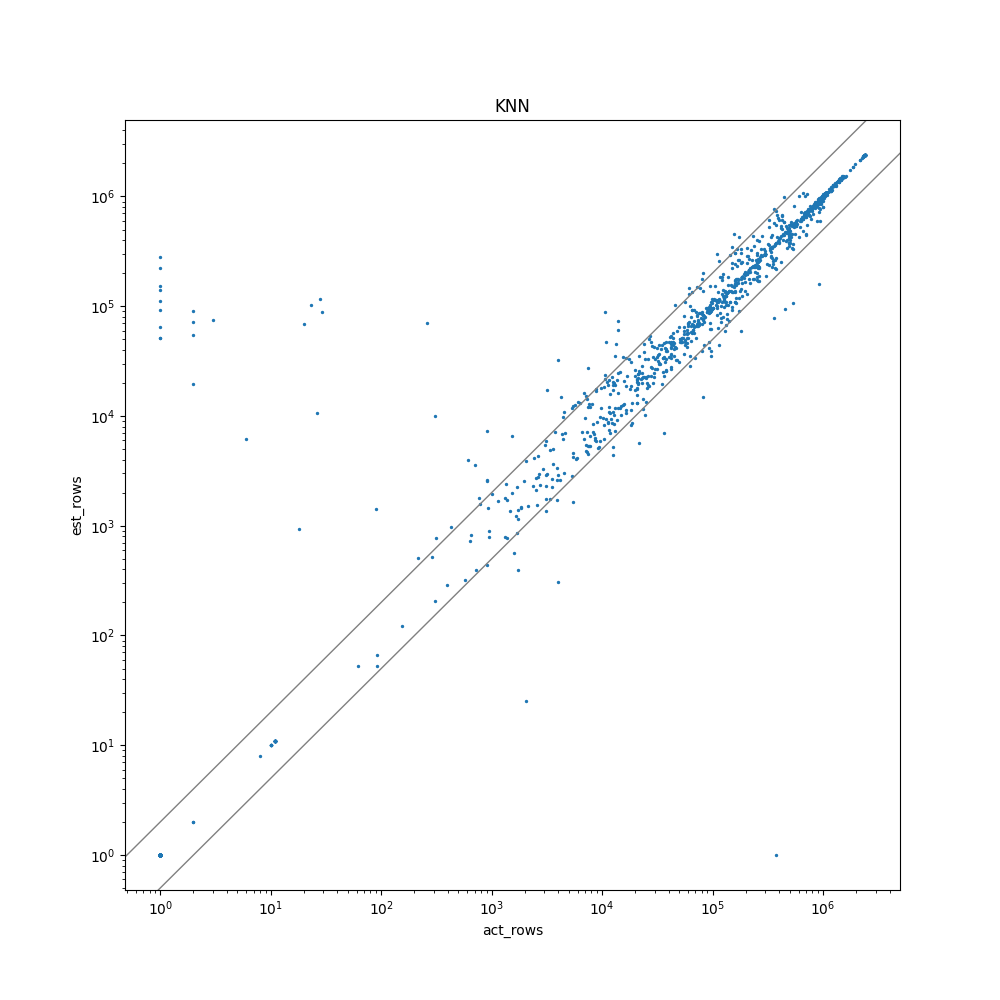
1. EBO方式：



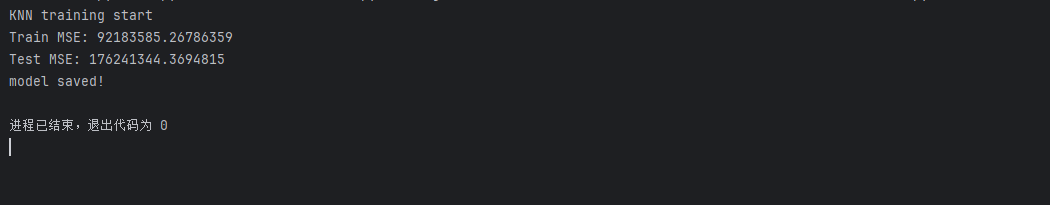
1. MinSel方式：

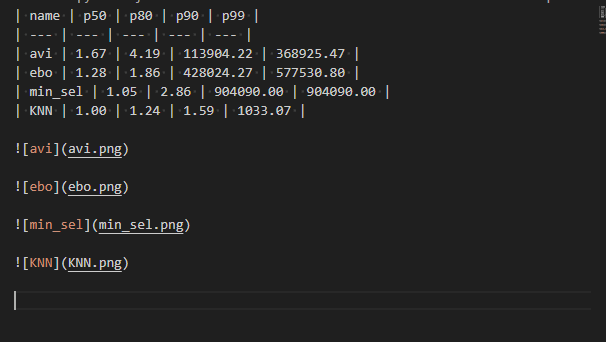


1. KNN：

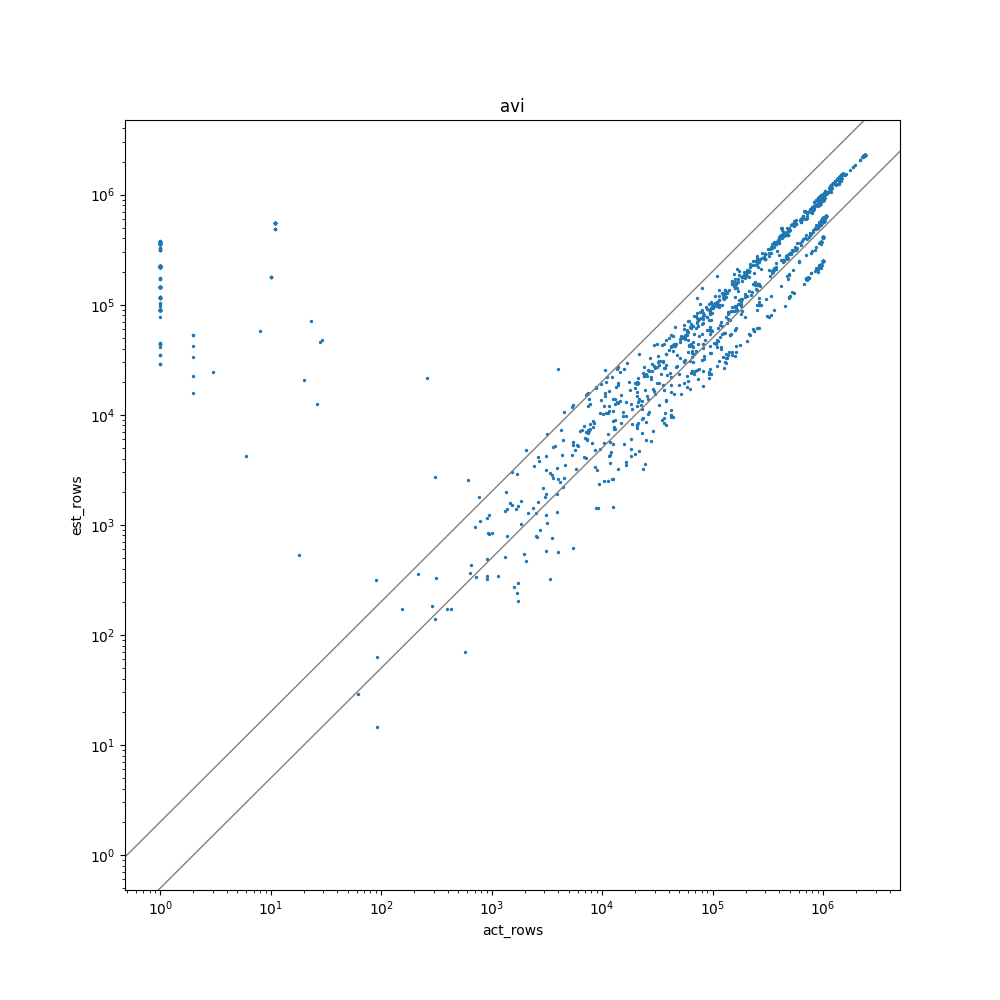


1. 运行截屏：

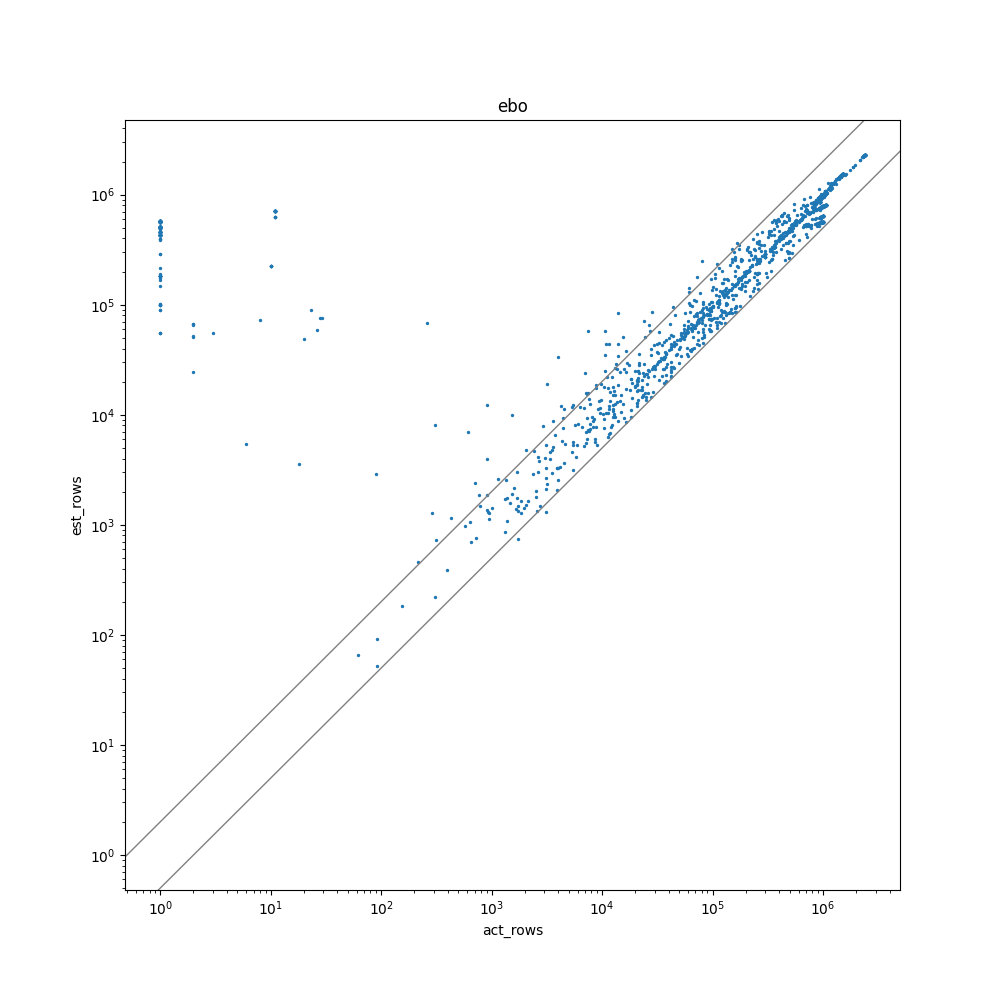




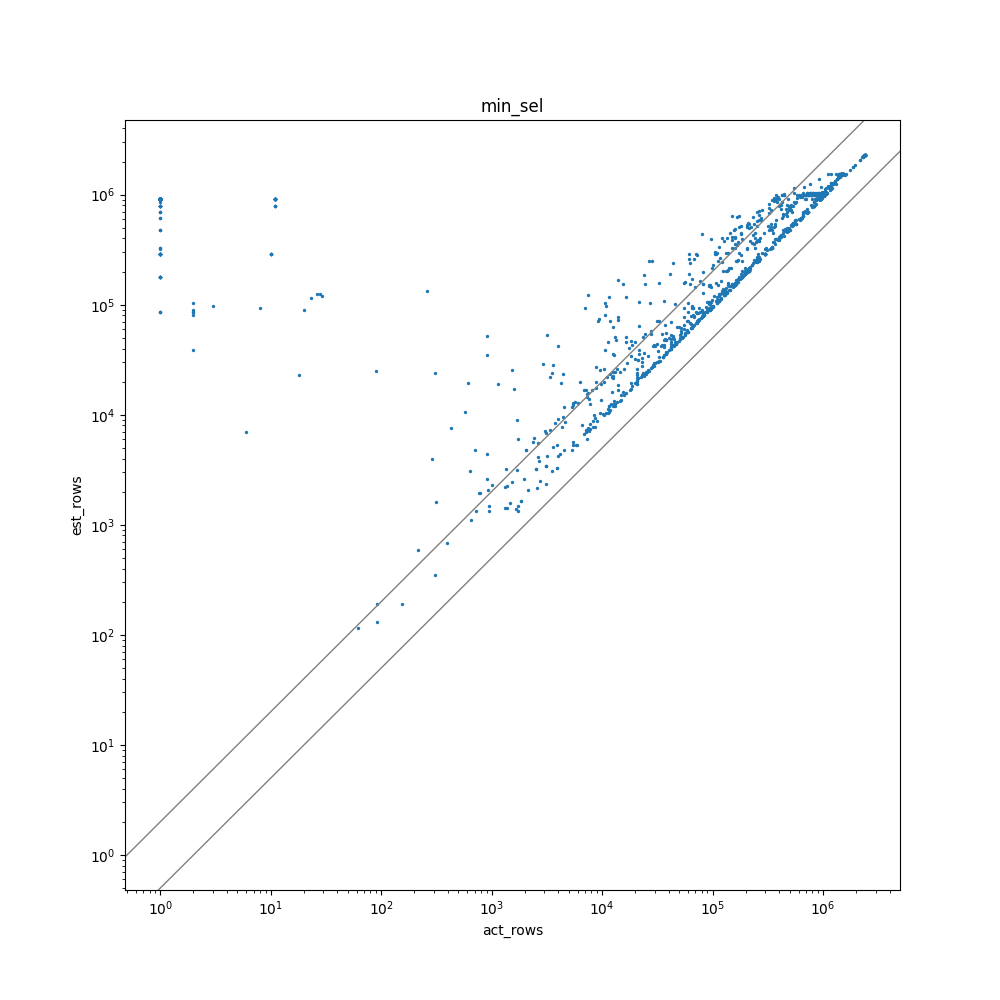
1. Lightbgm模型：
2. AVI方式：



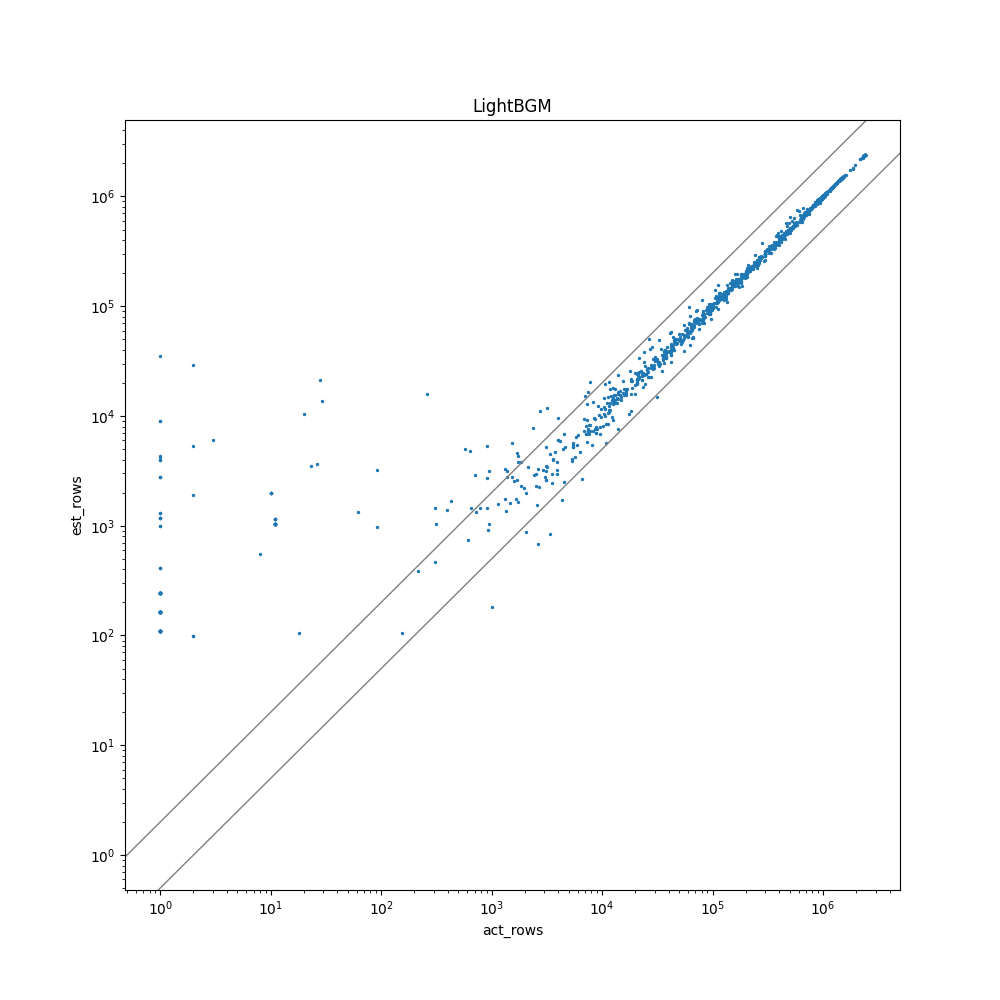
1. EBO方式：



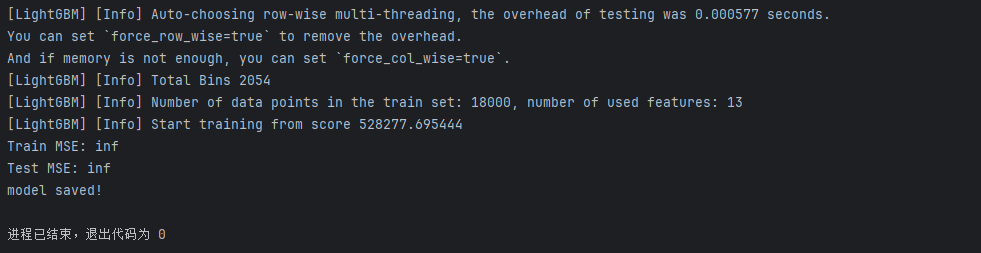
1. MinSel方式：

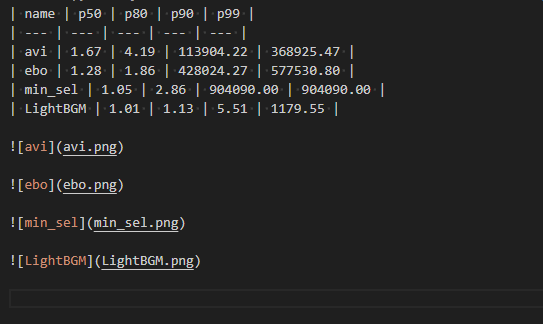


1. Lightbgm：



1. 运行截屏：





1. EBO方法和MinSel方法的实现：

（1）EBO方法补全：

Exponential BackOff (EBO):此方法不同AVI方法考虑所有的列，其认为应选出4个最小的选择率组成总的选择率，结合得到如下的式子:

IMG_256

其中代表第k小的选择率。通过将总的选择率乘上数据总行数，便得出EBO方法估计出来的基数。

实现代码：

class ExpBackoffEstimator:

"""

Exponential BackOff: When columns have correlated values, AVI assumption could cause significant underestimations.

Microsoft SQL Server introduced an alternative assumption, termed as Exponential BackOff, where combined selectivity is

calculated using only 4 most selective predicates with diminishing impact. That is, combined selectivity is given by

s(1) \* s(2)^(1/2) \* s(3)^(1/4) \* s(4)^(1/8),

where s(k) represents k-th most selective fraction across all predicates.

"""

@staticmethod

def estimate(range\_query, table\_stats):

selectivities = []

# Calculate selectivity for each column in the range query

for col in range\_query.column\_names():

min\_val = table\_stats.columns[col].min\_val()

max\_val = table\_stats.columns[col].max\_val()

(left, right) = range\_query.column\_range(col, min\_val, max\_val)

col\_cnt = table\_stats.columns[col].between\_row\_count(left + 1, right) # (left, right) -> [left, right)

col\_sel = col\_cnt / table\_stats.row\_count

selectivities.append(col\_sel)

# Sort selectivities in ascending order

selectivities.sort()

# Take the four most selective (smallest) values

top\_4 = selectivities[:4]

# Calculate combined selectivity using Exponential BackOff formula

combined\_sel = 1.0

for i, sel in enumerate(top\_4):

combined\_sel \*= sel \*\* (1 / (2 \*\* i))

return combined\_sel

实现思路：

（A）计算每列的选择率：

遍历查询条件中涉及的所有列。

通过 table\_stats 和查询范围 (left, right) 获取每列的选择率。

（B）排序选择率：

按选择率从小到大排序以找到最小的选择率。

（C）选择前4个最小选择率：

根据 Exponential BackOff 方法，仅使用前4个最小选择率。

（D）计算总选择率：

使用公式 s(1) \times s(2)^{1/2} \times s(3)^{1/4} \times s(4)^{1/8} 计算最终选择率。

（E）返回估算值：

将最终选择率乘以数据总行数得到估算的基数。

（2）MinSel方法补全：

Minimum Selectivity (MinSel):该方法较为简单，其认为总的选择率便是最小的选择率。通过将总的选择率乘上数据总行数，便得出MinSel方法估计出来的基数。

class MinSelEstimator:

"""

MinimumSel: returns the combined selectivity as the minimum selectivity across individual predicates

"""

@staticmethod

def estimate(range\_query, table\_stats):

min\_selectivity = float('inf') # Start with the maximum possible value

# Calculate selectivity for each column in the range query

for col in range\_query.column\_names():

min\_val = table\_stats.columns[col].min\_val()

max\_val = table\_stats.columns[col].max\_val()

(left, right) = range\_query.column\_range(col, min\_val, max\_val)

col\_cnt = table\_stats.columns[col].between\_row\_count(left + 1, right) # (left, right) -> [left, right)

col\_sel = col\_cnt / table\_stats.row\_count

# Update the minimum selectivity

min\_selectivity = min(min\_selectivity, col\_sel)

return min\_selectivity

实现思路：

（A）初始化最小选择率：

设定初始值为无限大，以确保后续计算中能够正确找到更小的选择率。

（B）计算每列的选择率：

遍历查询条件涉及的所有列。

使用列的 (left, right) 范围从 table\_stats 中获取行数。

计算选择率 col\\_sel=\frac{总行数}{col\\_cnt}

（C）更新最小选择率：

对每列选择率进行比较，保留最小值。

（D）返回最小选择率：

最小选择率即为估计的基数（未乘总行数），它表示满足查询条件的最少比例。