Distributed Postdiction

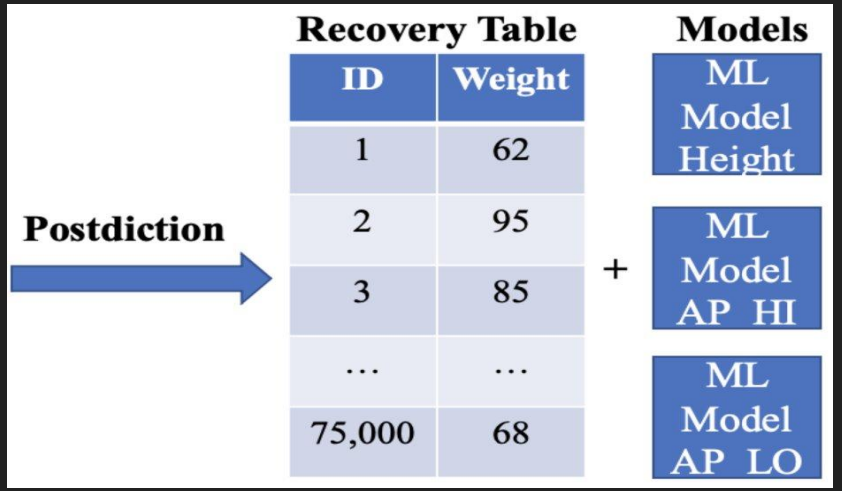
Trevor Petersen

What is Postdiction

- Prediction: aims to make a statement about the future value of a tuple
- Postdiction: aims to make a statement about the past value of a tuple which was deleted to free up space [IEEE MDM `18]

Original Table

ID	Height	Weight	AP_HI	AP_LO
1	168	62	110	80
2	181	95	130	90
3	156	85	140	90
…	…	…	…	…
75,000	164	68	110	60

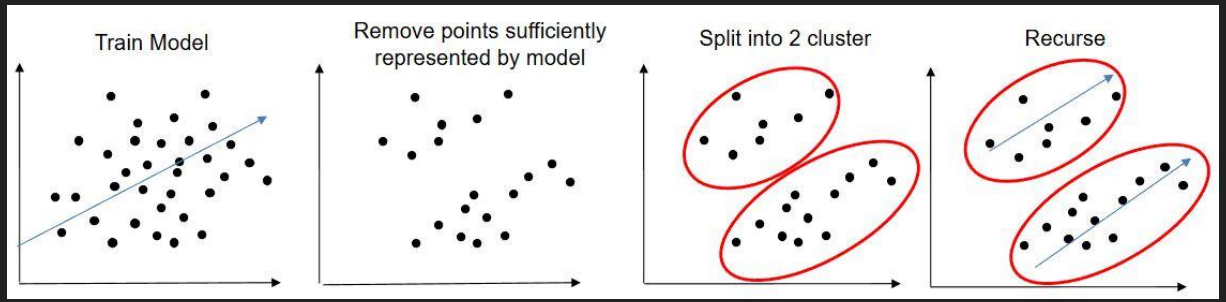


Goals of the Project

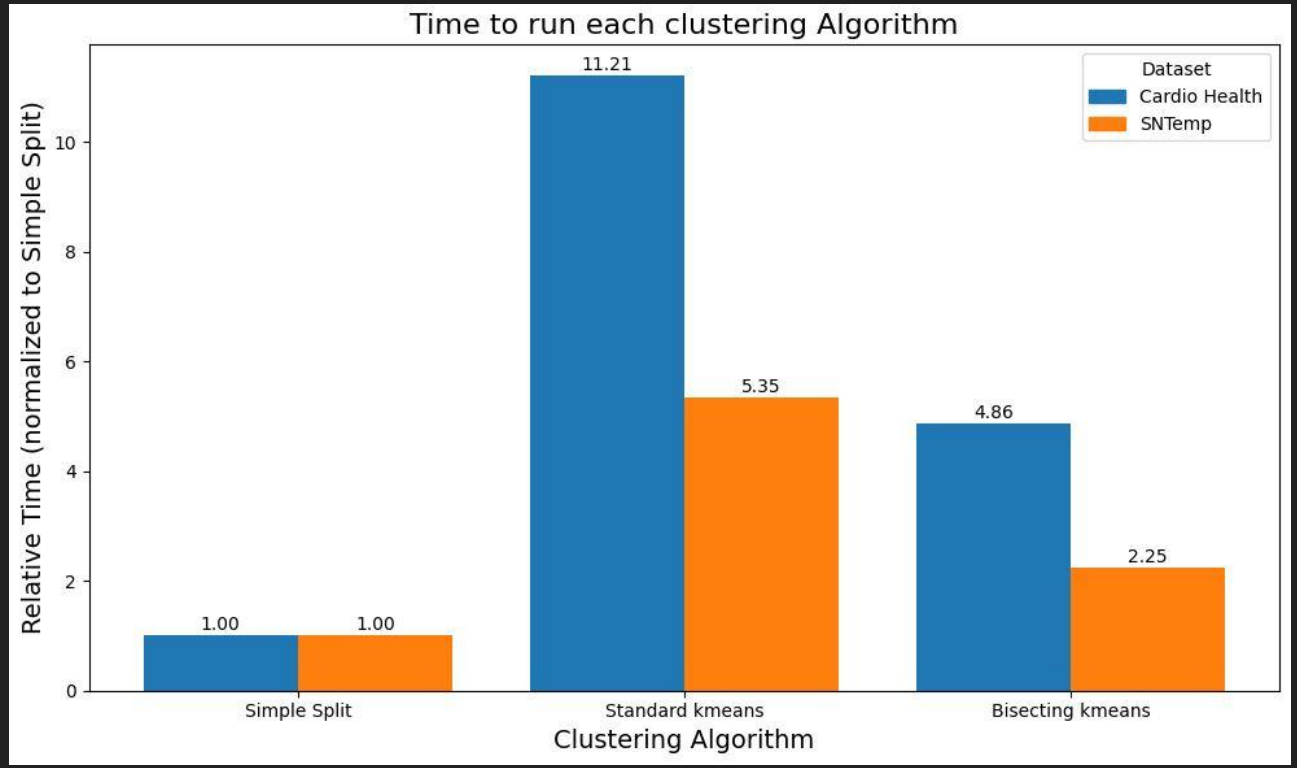
- Address long training times of postdiction while still not violating our error threshold
- 1. Create a fitting method that can quickly assign points into existing models.
  - In this scheme, postdiction would be split into two phases, a training phase and a fitting phase. The training phase would create the models and the fitting phase would then place all new values into these existing models
- 2. Create a Federated learning method to expand the training size used to create the models.
  - Goal is to get more training data to create models that represent all of the data before going to fitting.

An important change

- Previously, in training and split approach used the clustering technique kmeans
- We changed it to just use the relative position of the points in relation the last model trained



The Difference



Fitting data

Method 1 - Binary Search

Steps:

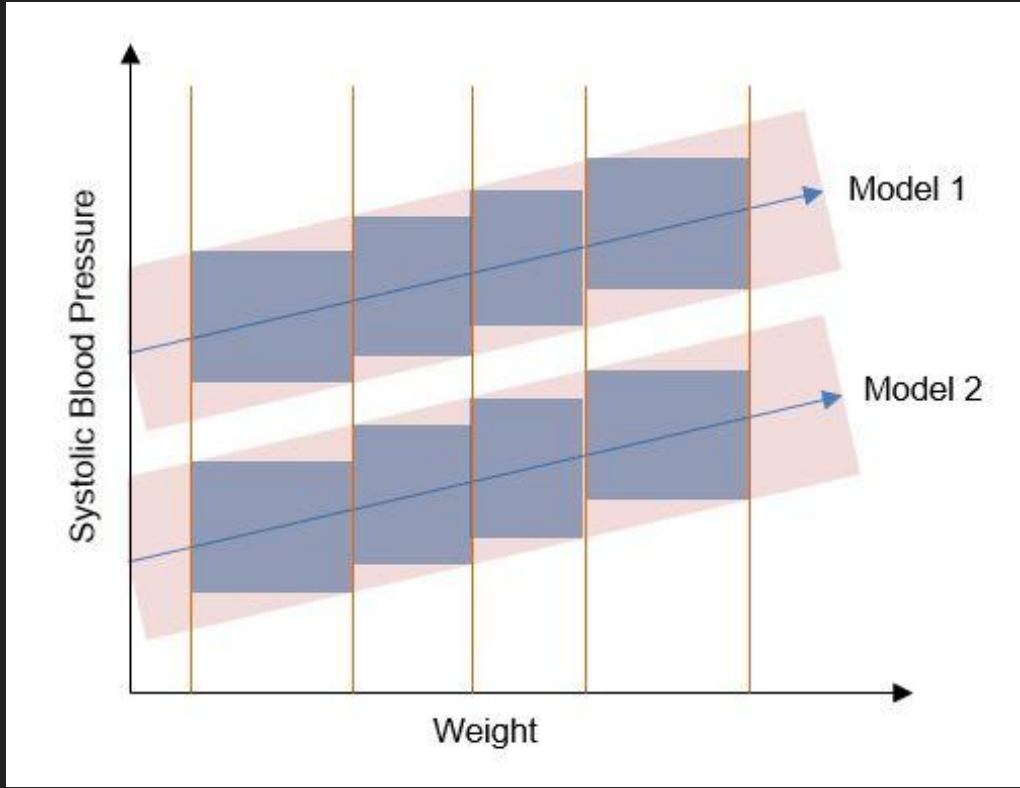
1. Sort all models based on y-axis value. Use y-axis as a prediction point for a model. *sorted_y_axis_values*=[$m_0(0)$, …, $m_n(0)$]
2. given a new point (x,y) perform binary search on *sorted_y_axis_values* to find the model m_i the y most closely aligns to
3. Test (x,y) on model m_i and see if its result is with the program argument error constraint c
4. If it is outside the error threshold , go to the next closest model based on if you were above or below the model
    a. m_{i+1} if above and m_{i-1} if below
5. Continue until you find a match, a gap, or hit the end of the list

Method 2 - Interval Trees

Steps:

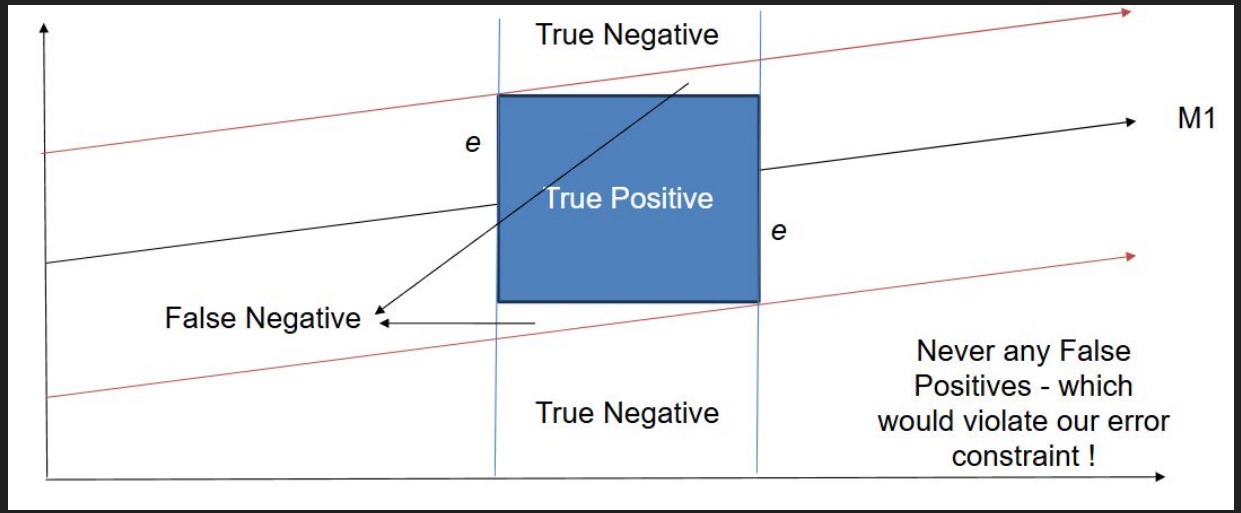
1. Create an interval tree that breaks down the x-axis into a continuous set of intervals: x_intervals = [(*min(x),1*), (1, 3), … , (*70, max(x)*)
2. For every *x_interval* in *x_intervals* create an interval tree *y_interval_tree* which represents the valid bounds for every model *m* within that x interval.
3. Within an *x_interval* (x_1, x_2), with a program error threshold of *e*, a model *m* has the bounds (y_1, y_2) determined as follows:
    - Negative slope:
        - $y_1 = m(x_1) * (1 - e)$
        - $y_2 = m(x_2) * (1 + e)$
    - Positive slope:
        - $y_1 = m(x_2) * (1 - e)$
        - $y_2 = m(x_1) * (1 + e)$

Method 2 (Cont.)



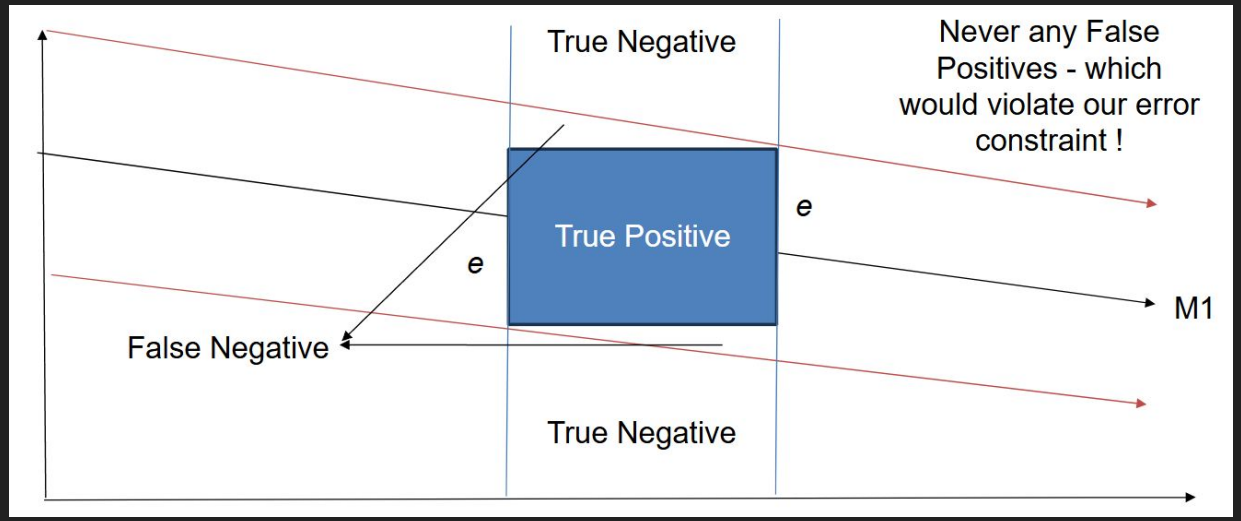
Method 2 - Positive Slope

- Positive slope:
  - $y_1 = m(x_2) * (1 - e)$
  - $y_2 = m(x_1) * (1 + e)$



Method 2 - Negative Slope

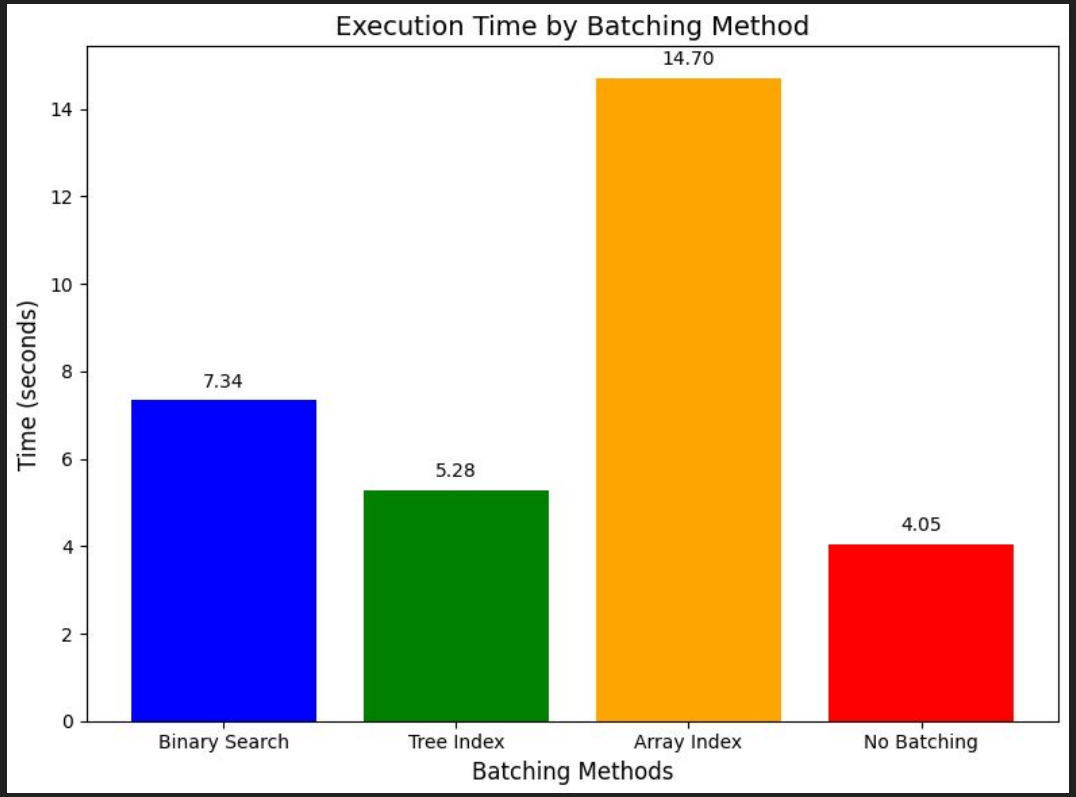
- Negative slope:
  - $y_1 = m(x_1) * (1 - e)$
  - $y_2 = m(x_2) * (1 + e)$



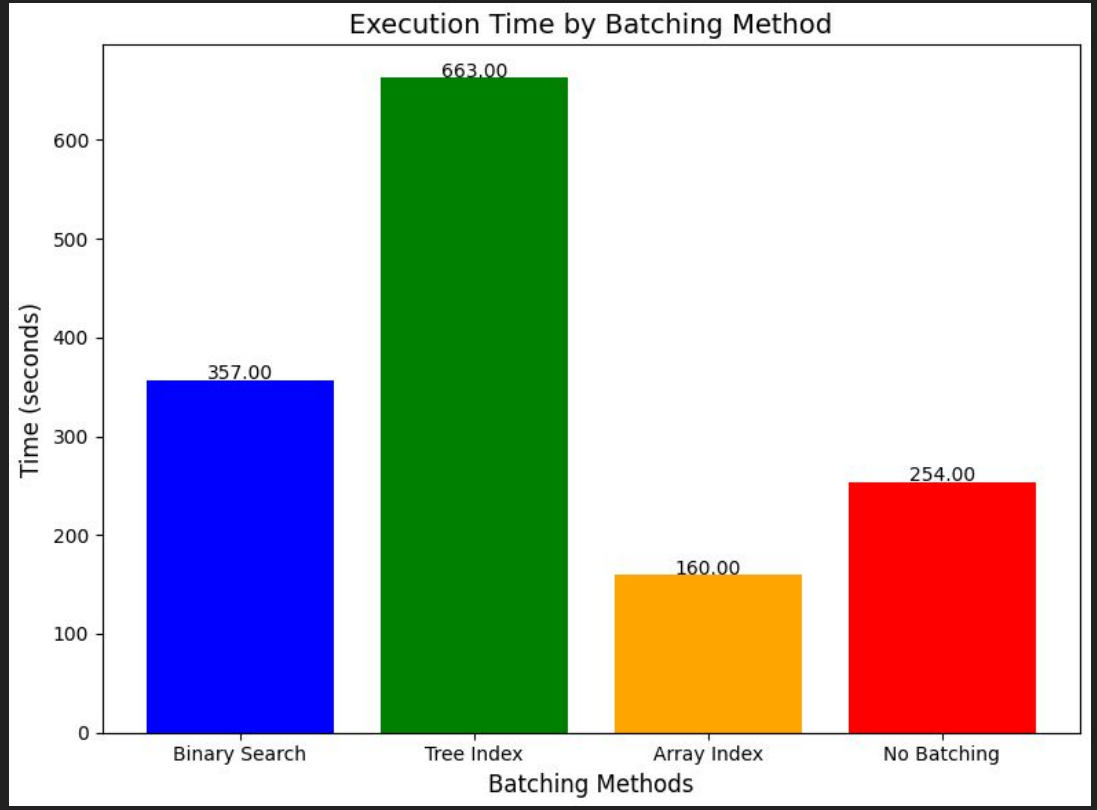
Method 2a - Interval Arrays

- Use an array like implementation to condense information onto fewer pages for better locality and performance
- x interval tree formatted as
  - *x_interval_array* = [*min(x), … ,max(x)*]
  - *y_interval_tree_ptrs* = [y_tree_1 , … , y_tree_s] where *s* = number of splits
- Any given y interval tree formatted as (where *n* = number of models):
  - *y_min_array* = [min_y_1 , … , min_y_n]
  - *y_max_array* = [max_y_1 , … , max_y_n]
  - *model_ptrs* = [m_1 , … , m_n]

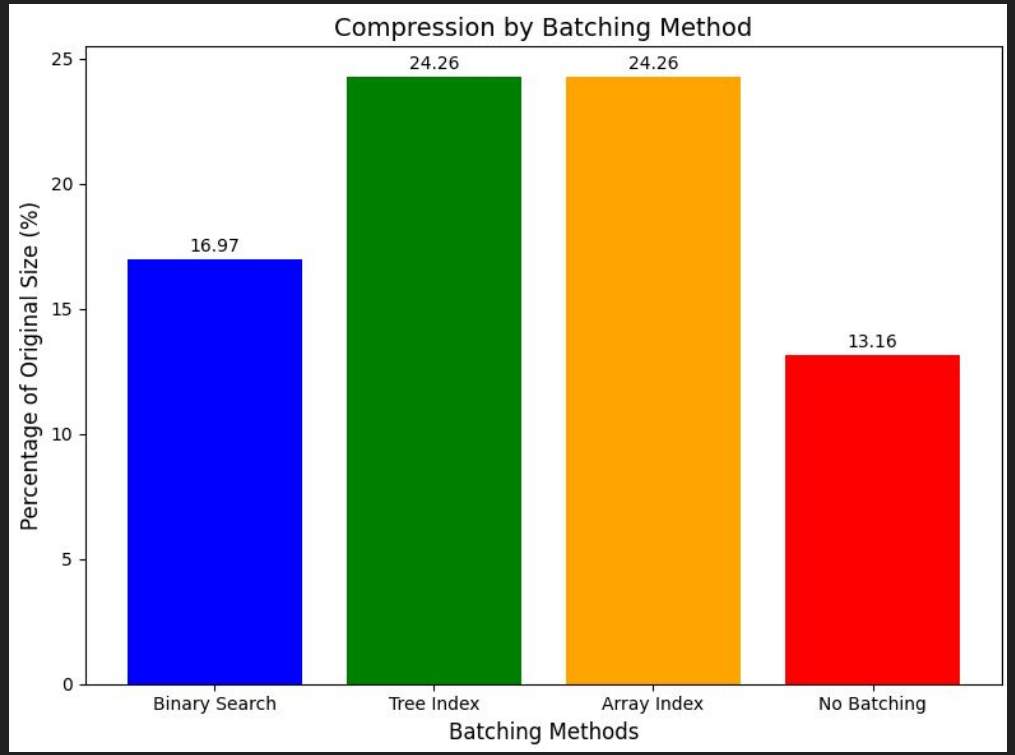
Insights from fitting data (cardio - 2.8MB)



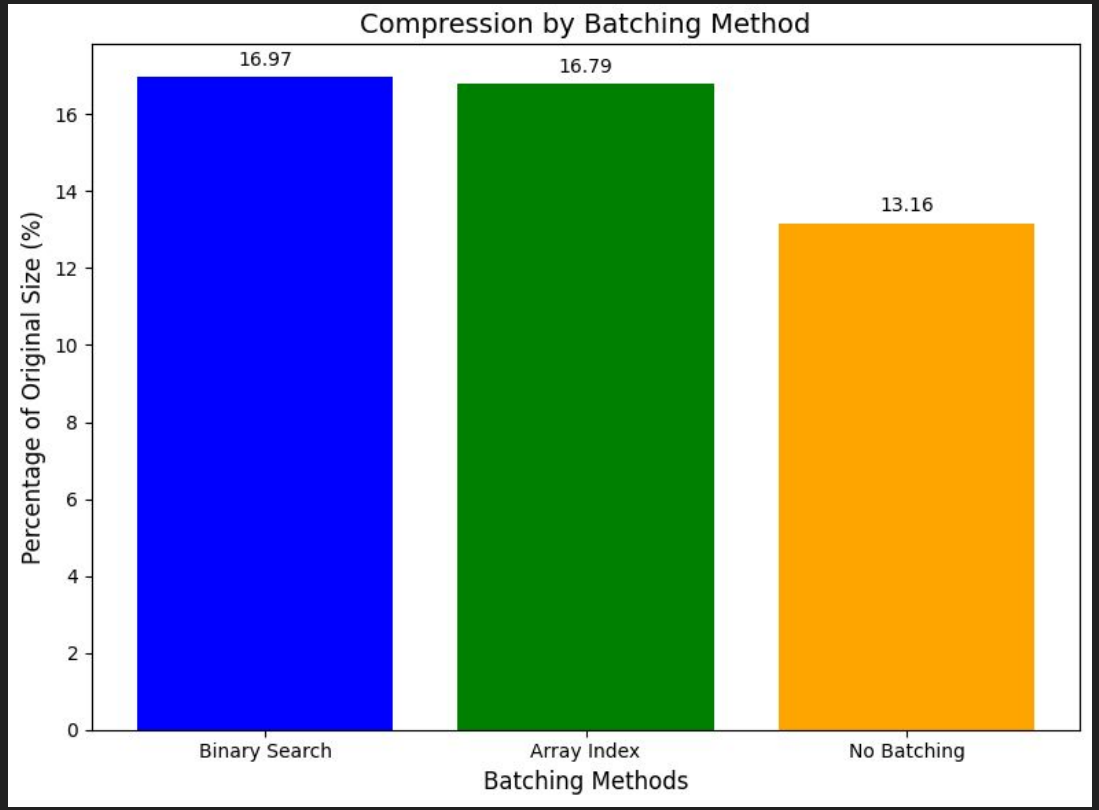
Insights from fitting data (sntemp - 166MB)



How batching affects accuracy (sntemp)



Adjusted for my coding error

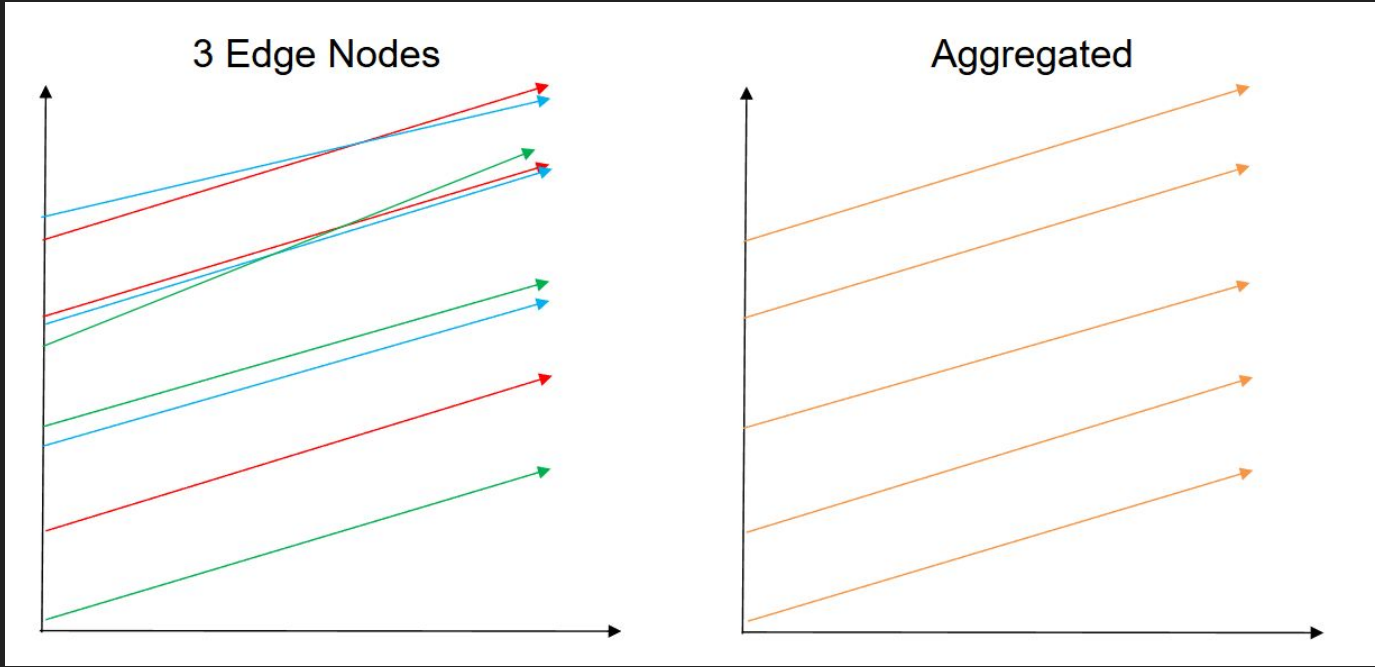


Federated Learning

What federated learning looks like in Postdiction

- One Idea: Using a lower error tolerance on the edge nodes and having models that have chunks (ranges on the x-axis) that would fit entirely into one model on the central node…
- Have many nodes train models normally, and then perform an analysis on all models and select a set that represents all the data
  - Remove extra models that are extremely similar. Keep the unique models from the different edge nodes
  - Helps to avoid undersampling the data
  - Would have to refit all data into the models at the end, which is is unlike traditional federated learning, where each edge node operates on its own shard of data (and the central node only takes in their updates).

Visual Representation



Questions