Pattern Mining Datasets to determine Robustness Classification

Joshua Huang
joh009@ucsd.edu

Marlon Garay
mjgaray@ucsd.edu

Babak Salimi
bsalimi@ucsd.edu

Abstract

Data robustness is the ability of a dataset to remain clean and reliable for data tasks in the face or uncertainty and errors. As such, a dataset's robustness becomes highly important in its reliability. However, there isn't a simple way to directly confirm a dataset's robustness in comparison to others in a cost and time efficient manner. That is where ZORRO and pattern mining approaches come in. The utilization of these two techniques lets us discover scenarios of the worst case uncertainty for a given dataset and utilize them to confirm a measurement of value that would represent just how robust a given dataset is along with allowing us to compare measurements of robustness between different datasets to get a sense of the difference in sensitivity across all of our chosen samples. The goal of this paper is to demonstrate that dataset robustness can represented in a value that's generated in both a time and cost efficient manner and that said value not only gives us an idea of the level of robustness for a given dataset but also a comparison of values towards other similar dataset robustness values creating differences and thus conclusions that certain datasets are more reliable in the face of errors and uncertainty in comparison to others.

Website:
Code: https://github.com/rx-72/Dataset-Robustness-Classification-and-Error-Testing

1 Introduction

In our previous paper (Marlon Garay (2024)), we demonstrated the importance of robustness in datasets along with providing a method to accurate get a value that could be utilized for measurement. While effective, this method could have been improved upon and didn't full tap into the true underlying representation of the dataset. We also didn't have a great way of comparing these values across different datasets since the parameters utilized (robustness radius, uncertainty radius, etc.) often differed from one dataset to the next. As such, we'll be aiming to introduce a new method to capture the dataset's important values and demonstrate the its robustness by targeting these values as the worst case scenario for error injection. Once again, our solution utilizes system from ZORRO (can referred to at: Zhu et al. (2024)) to determine the descent of robustness given larger and larger data uncertainty. Our new method will have a larger focus on Gopher (2024) by utilizing pattern mining techniques to learn all possible patterns of a dataset and gain knowledge of the underlying representation (and in turn creating more effective error injection techniques.) The goal is similar to our previous paper, the creation of a robustness dataset classifier that can measure robustness and compare it across different datasets. To accomplish this, we must learn the patterns to a dataset but in a cost efficient manner and in a way that doesn't conflict across different datasets. We believe we've discovered a method that accomplishes both tasks along with being a major improvement over our previous paper's methods. Such a new method is able to achieve a strong pattern representation for a given dataset which in turns allowed us to create much more effective error injections and robustness measures in comparison to last quarter.

2 Methods and Datasets

We'll the following baseline for our classifier:

1. Train-test split the data that we want to gain a robustness measure on (we use a configuration of 80 percent train, 20 percent test here).

2. Inject a specified percentage of targeted points in the y labels of the train data with a given distribution of a chosen error value.

3. Utilize a chosen method on the train data to try and address the error injected. Here, we have two options between utilizing Meyer (parameter intervals) or ZORRO. In general however, ZORRO performs much better at robustness retaining than Meyer, so we'll base more of our results upon it.

4. After using a method to attempt to clean the error injection, check the results after utilizing the train data to fit the test data. This result is given as a numerical value that represents itself as a robustness ratio, where a ratio of 100 implies the data is very robust and vice versa.

5. We use a heat map and line plots to demonstrate our findings of robustness using this process under different parameters of ”percentage of data missing” and ”distribution width of uncertainty injected”.

The main method we'll utilize to create effective error injections is our heuristic pattern mining approach. We'll go into it in further detail below in section 2.1 but for a brief description, we capture all possible predicates and inequalities for a given dataset before slowly filtering out obsolete ones based on given patterns we're aiming for.

We'll also go into further detail of the parameters that are tuned per each dataset in section 2.2, along with going over how we demonstrate our results in section 2.3 In section 2.4, we'll showcase the datasets we utilized for our testing along with their intricacies and we introduce our normalization method in section 2.5 for comparing robustness scores across different datasets.

2.1 Heuristic Pattern Mining Influence Analysis

To reiterate, the idea is as follows:

1. There exists a multitude of possible patterns for each feature in a chosen dataset that can represent a potential sample size. (For instance, a temperature value of less than 100 degrees Celsius may represent 50 percent of the dataset)

2. Based on this idea, we grab all possible single predicate patterns that follow such a definition based on the number of unique features each column in the dataset has before filtering the dataset repeatedly with each unique singular pattern. (From the previous example, ”100” is a unique value in the temperature column thus creating 3 possible inequalities ”>”, ”=”, ”<”. We create these 3 inequalities per each unique value in a column per each column. So we'll have $3*n_k*k$ where n_k is the number of unique values per column, and k is the number of columns. We may bin the values certain columns should there be too many unique values)

3. After all single predicate patterns are captured, we combine these patterns together to create combinations. For instance, we may combine a pattern from one column (temperature $>$ 100) with another (area $<$ 50) to create on singular pattern combination (temp $>$ 100 $\wedge$ area $<$ 50). Note we do not combine patterns of the same column (so we will not combine two patterns that come from temperature) and that the maximum patterns combined is the maximum amount of columns (so if the number of columns is 7, the largest combination of patterns we can create is 7). That said, we tend to focus on patterns that combine on 2 or 3 unique columns since these combinations

4. Once all unique combinations of size ”k” are made (where k is some number such as 2, 3 , etc.), we start to filter the combinations based on certain utilized statistics, such as increase in mean squared error/mean absolute error or the correlation to the target variable (”y train”). This helps to filter across all possible patterns such

that we can find the ones that prove most effective at representing the dataset.

5. Once the list of combinations has been filtered to a reasonable size, we run a robustness test on each combination of patterns using fixed parameters. These parameters represent the realistic worst case scenario for a given dataset. The patterns that return the lowest robustness ratio under these parameters are the ones chosen as the most effective patterns for representing the data and for causing the worst label error in the dataset.

After finding the top patterns for representation and error injection, we'll run our ZORRO tests using the indexes that these patterns capture for further analysis. We also follow this up by compare results we'll receive to the ones from previous methods, such as our naive method or our leave one out approach from our previous paper (Marlon Garay (2024)). While pattern mining ends up taking much longer in terms of finding the indexes to target for maximum error, we'll soon see it ends up much more of an effective approach than our previous methods, especially on datasets that prove to be less sensitive to error injection.

2.2 Tuning Parameters

To find specific patterns of importance, we'll need to tune different parameters in the ZORRO model to capture a more specific representation. This differs from our previous study where our methods weren't as deliberate enough to require such a tuning. The parameters are as follows:

1. Uncertainty radius: This refers the percentage of uncertainty that is injected to all targeted indices. This can range from a number starting near 0 to 1.

2. Robustness radius: This refers to the label range where a point would be considered robust. For example if a point with a value of y was injected with some percentage of error based on the uncertainty radius and became a value of z, the point would still be considered robust if the value z fell in the range of [y - 'robustness radius', y + 'robustness radius']. Since robustness radius is correlated with uncertainty radius, we tend to keep uncertainty radius consistent across all dataset tests, so we can clarify the differences in robustness radius.

3. Uncertainty Injection Percentage: The maximum percentage of the dataset that gets injected with error. In relation to our pattern approach, if the percentage given is for example 10 percent or 0.1, all patterns of size k that capture 10 percent or less of the full dataset are utilized for the process in finding the best index representation and error injection of the dataset. In turn, this allows us to find the best patterns that are, continuing with our example, 10 percent or less of the dataset that best represent the data and completely diminishes its robustness when indexes that fall under said patterns are error injected. This variable tends to be low, since lower percentages of important representations are much more interesting than large ones.

2.3 Result Representations

As with last time, we rely on utilizing heatmaps to visualize the robustness ratios across the different pattern injections we'll be using. Each ratio gets calculated from the average and standard deviation of multiple runs across different random seeds before being converted to percentage decimals for ease of readability. Since we're more interested in the worst case scenarios however, we tend to rely more on ZORRO than Meyer methods since it has better robustness management. As such, our results may not overly cover the Meyer portion results of data in comparison to the ZORRO results.

We'll also utilize line plots in addition to our result representation. This is because at times we may choose to focus on results that are kept stagnant rather than constantly changing. This is in particular regarding the variable "Uncertainty Injection Percentage", as this time we plan to use all target indices that would fall under a certain pattern for error injection. This is to ensure consistency in the results as we want to see how much a pattern may cause robustness diminishment with retrospect to the amount of data it captures from the dataset. The lineplots in questions will showcase the robustness under a given pattern error injection undergoing a diminishment as the inputted uncertainty radius would increase in percentage size.

2.4 Datasets

For our tests, we utilize the below 4 datasets, 2 being the same ones from our previous paper (Marlon Garay (2024)) and 2 new datasets chosen to expand upon the potential range of data sensitivity shared across all of our sets

1. Insurance: A dataset focused on predicting the amount of charges for insurance for each specific patient. It contains high sensitivity between its features and the prediction labels. A display of the dataset head is provided below:

age	bmi	children
53.000000	26.600000	0.000000
53.000000	21.400000	1.000000
18.000000	37.290000	0.000000
60.000000	24.035000	0.000000
45.000000	33.700000	1.000000

2. MPG: A dataset focused on predicting miles per gallon per each given vehicle. It contains low sensitivity between its features and the prediction labels. A display of the dataset head is provided below (Note only the first 4 columns are showcased to prevent overflow):

cylinders	displacement	horsepower	weight
8.000000	350.000000	125.000000	3900.000000
8.000000	455.000000	225.000000	3086.000000
4.000000	91.000000	68.000000	2025.000000
4.000000	122.000000	86.000000	2226.000000
4.000000	97.000000	67.000000	2065.000000

3. Fire: A dataset focused on predicting the area covered by a fire given specific fire conditions. It contains medium sensitivity between its features and the prediction labels. A display of the dataset head is provided below (Note only the first 4 columns are showcased to prevent overflow):

X	Y	FFMC	DMC
3	5	93.500000	139.400000
4	5	92.900000	133.300000
3	4	93.000000	75.300000
6	5	75.100000	4.400000
4	4	94.800000	108.300000

4. Boston: A dataset focused on predicting the value of owner's homes in Boston given specific parameters about the house itself and the area in question. It contains low sensitivity between its features and the prediction labels. A display of the dataset head is provided below (Note only the first 4 columns are showcased to prevent overflow):

CRIM	ZN	INDUS	CHAS
0.141500	0.000000	6.910000	0
0.154450	25.000000	5.130000	0
16.811800	0.000000	18.100000	0
0.056460	0.000000	12.830000	0
8.792120	0.000000	18.100000	0

Based on the sensitivity of each dataset, we should expect in our tests lower robustness on Insurance, followed by medium sized robustness with Fire and high robustness by both Boston and MPG equally.

2.5 Normalization

While our pattern and target indices methods allow us to find strong robustness error injection methods, we don't have an effective method to compare ratios across these different

methods and datasets. For instance how do we know our pattern mining results are truly more effective than our leave one out results when parameters between the two may be different? Or if the Insurance dataset is more or less robust than the MPG dataset? This is solved by usage of our normalization method.

We'll explain comparing methods on a single dataset first since it's simpler. Suppose we had a set percentage uncertainty to inject upon (say 10 percent) and a set radius of uncertainty range that we could possibly add to a label for error injection (say plus or minus the difference between the max label value and the min label value times a constant like 0.25). In these set conditions, all methods are thus the same except for what target indices they may utilize for error injection. To truly get a sense of which is the best, we'll constantly our robustness calculator with the aforementioned conditions against each dataset trying to hit a specific threshold of ratio of robustness (say 0.5). The key factor here is the robustness radius. When we first run our robustness function on a method, the robustness radius will be set at a constant, say 1. Than if the function fails to return a ratio greater than 0.5 (our threshold), we'll increment the constant by a bit (say by 0.01 for example). We then repeat the process until the function pasts the threshold, which we then record the value of our robustness radius at this point that pushed the utilized method to reach such a robustness ratio. We then repeat this for process on every method that we are testing on. By the end we generate score based on all ratios that we've received for each method on a single dataset. The method with the lowest score is the method that had the most effective error injection and whose target indices could be considered the worst case scenario for that dataset.

For comparing multiple datasets, we gather multiple datasets and repeat a similar process to the single dataset. The only catch here is that for each dataset, the methods related to that dataset need to be normalized with respect to the range of their y values. For example, the insurance dataset has a label range for min and max of (1121.8739, 63770.42801) vs the MPG dataset which has a label range for min and max of (9.0, 46.6). All of our scores are adjusted with respect to their datasets' label ranges before comparing again. If we choose to use only the worst case scenario method per each dataset, we can then conclude that the dataset with the lowest score is the dataset that is the least robust out of the given datasets with respect to its task, while the dataset with the largest score is the dataset that is the most robust out of the given datasets with respect to its task.

3 Results

3.1 Pattern Mining Results (Flawed)

Below we contained each of the error injection for the given patterns found per dataset and displayed them in a similar format to what we did in Marlon Garay (2024). The heatmaps represent the diminishing of robustness under certain uncertainty ratios and specific percentage of dataset error injections, similar to our previous paper. We've displayed each of the heatmaps created for the top 3 best error injections patterns found per dataset and compare them against the naive method. Note we'll also include the specific pattern found

in conjunction with the dataset. Each pattern has also been checked to ensure no overlap across two different patterns. Note also the heatmaps are flawed for uncertainty percentages below 10 percent as there is no inherent order of importance in how we inject errors in regards to pattern mining. We still include them regardless as their error injections are still clearly more effective than our previous methods. A more accurate representation will be provided in the next subsection.

3.1.1 MPG dataset

The specific best error injection patterns found via pattern mining are as follows:

```
Pattern 1:
'weight' < 3376.5
'acceleration' < 12.122222222222222
'model year' < 82.0

Pattern 2:
'displacement' > 69.615
'weight' > 2274.3125
'origin' > 1.0

Pattern 3:
'weight' > 2494.75
'acceleration' > 10.311111111111112
'model year' > 78.0
```

Below are our pattern mining results running on the heatmap, represented in figure 1. As you can see, pattern 1 has nearly twice the more effective error injection vs the naive approach under the worst case scenarios (0.25 uncertainty radius and 0.1 percentage of dataset to inject on), where the ratio is almost half of what it is in the naive method. Meanwhile, pattern 2 and pattern 3 are close behind it and also much more effective than the naive method too. Vs the discretization histogram method and the leave one out methods from Marlon Garay (2024), the worst case scenarios with given models gave us 41.2 and 42.5 respectively as our worst case scenarios and yet, our pattern approach here was able to discover further gap in error injection effectiveness by mining the patterns from the MPG dataset.

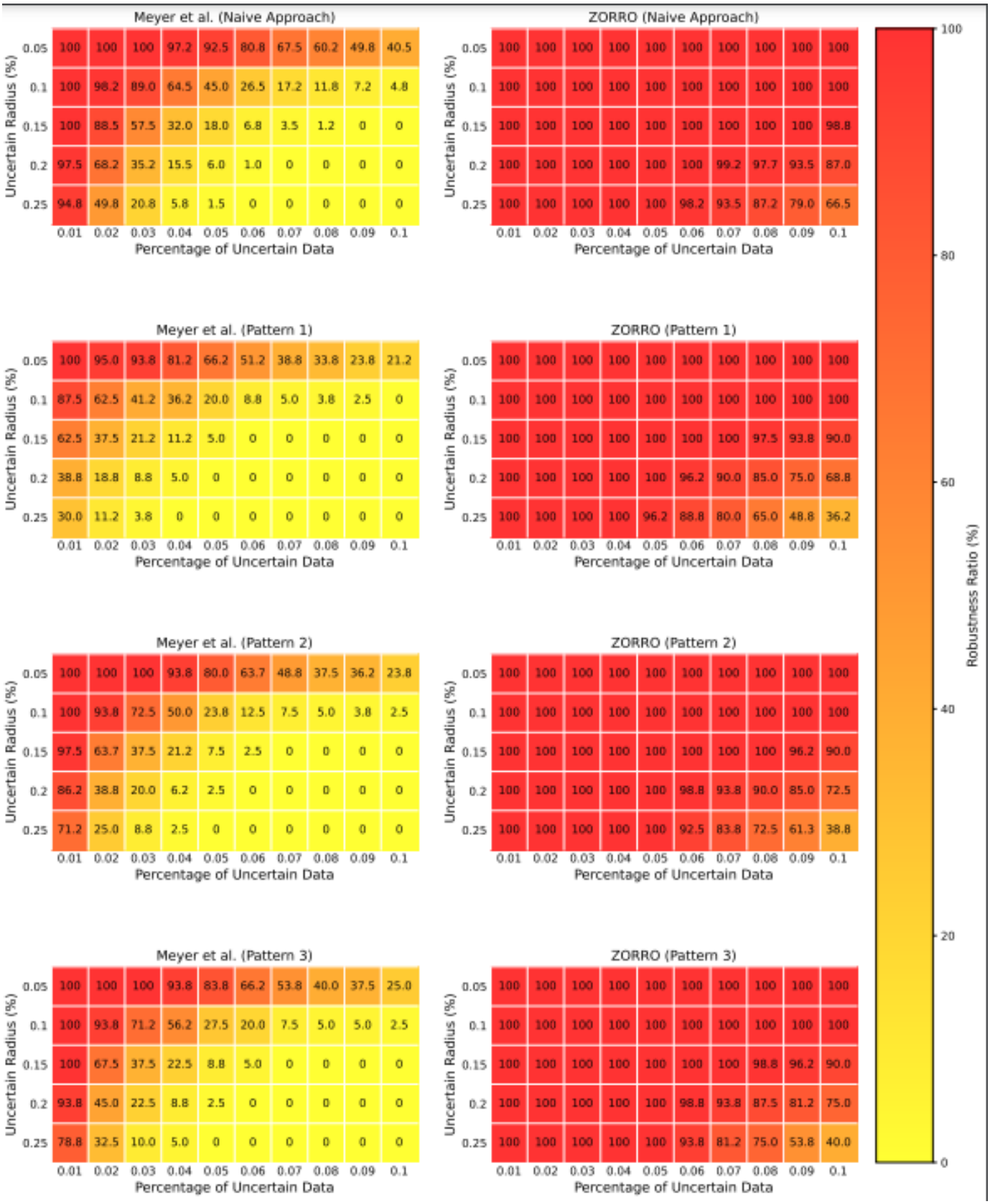

Figure 1: MPG Pattern Mining Heatmap

3.1.2 INS dataset

The specific best error injection patterns found via pattern mining are as follows:

```
Pattern 1:
29.476363636363637 < 'Body Mass Index' < 31.165909090909093

Pattern 2:
'Body Mass Index' < 21.028636363636366

Pattern 3:
31.165909090909093 < 'Body Mass Index' < 32.85545454545455
```

Below are our pattern mining results running on the heatmap, represented in figure 2.

Notice here that our pattern mining methods were less effective in this scenario. For instance, the naive method reached a ratio of 2.7 under the worst case scenarios given here, but all 3 of our patterns that we determined were the best for representing 10 percent of the data gave ratios that were still larger than the naive method. In fact, the decrement in all 3 patterns seems far slower paced in comparison to what the naive method has, and the worse case ratio for these patterns is almost or above 4 times as large the worse case ratio of the naive method.

If we look at our Leave One Out methods and histogram methods on this dataset, we can find that the error injections there were much better than the ones on here. The Leave one out method for example had a robustness ratio of 1.5 as the worst case while the heuristic histogram had the worst case of 2.6 . Both methods were much more effective than pattern mining on the insurance dataset, which allows us to conclude that here pattern mining's effectiveness is more so reliant on how stable the dataset is. In our previous paper (Marlon Garay (2024)), we noted that the insurance dataset had a high sensitivity towards its labels (we'll prove this in our normalization results) and pattern mining is more focused on finding patterns that are good representations towards the labels. This may not be as effective if there's sensitivity between the labels and specific feature patterns, hence why pattern mining ends up the worse here vs the other 3 methods.

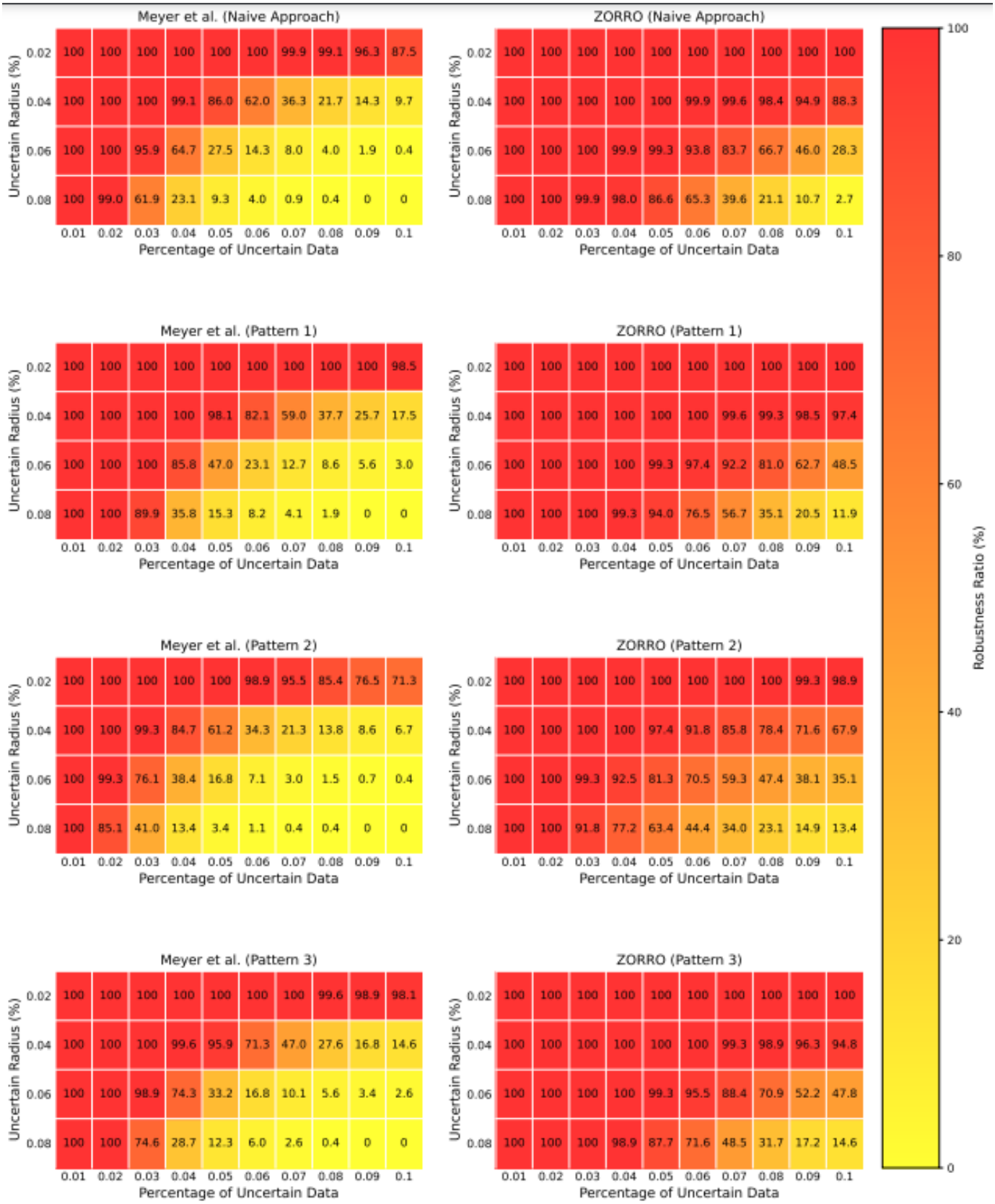

Figure 2: Ins. Pattern Mining Heatmap

3.1.3 Fire dataset

The specific best error injection patterns found via pattern mining are as follows:

```
Pattern 1:
11.76923076923077 < 'Temperature in Degrees Celsius' < 14.161538461538463
'Relative Humidity Percentage' < 68.125

Pattern 2:
'Relative Humidity Percentage' > 57.5
'Rain (mm/m2)' < 1.4

Pattern 3:
56.1 < 'ISI index from the FWI system' < -0.056100000000000004
'Wind speed (km/h)' < 4.5
```

Below are our pattern mining results running on the heatmap, represented in figure 3.

As we can see, Pattern 1 achieved a reduction in the robustness ratio vs the Naive method. Meanwhile Pattern 2 and Pattern 3 are also deemed more effective than the Naive method as well in fact getting pretty close to the same ratio as Pattern 1. Note the fire dataset is our second most sensitive dataset, though not nearly to the degree that the insurance dataset was. Despite that, pattern mining was still quite effective on this dataset here, which would imply that the effectiveness in terms of sensitivity may be highly reliant on a large amount of sensitivity (as in the case with the insurance dataset) as otherwise, pattern mining is a still a good error injection method (as we can see here with the fire dataset.)

Next we'll compare the leave one out and histogram methods. We don't have these on hand from our original paper but they can be directly viewed on our project website for reference. The most important fact is the worst case robustness ratio under given parameters and worst possible conditions. The leave one out worst case robustness ratio was 11.8 with linear regression and mean squared error. Meanwhile the histogram worst case robustness ratio was 19.2 also with linear regression and mean squared error. So here, the leave one approach is much more effective than the pattern mining approach, while the histogram approach is completely out done but still quite close. This matches up with our idea that sensitivity effects the performance of our pattern mining approach, where with a somewhat sensitive dataset, our pattern mining still remains more effective than the histogram and naive method but due to the amount of sensitivity in the labels on this dataset, it seems the leave one out approach ends up more effective on this scenario.

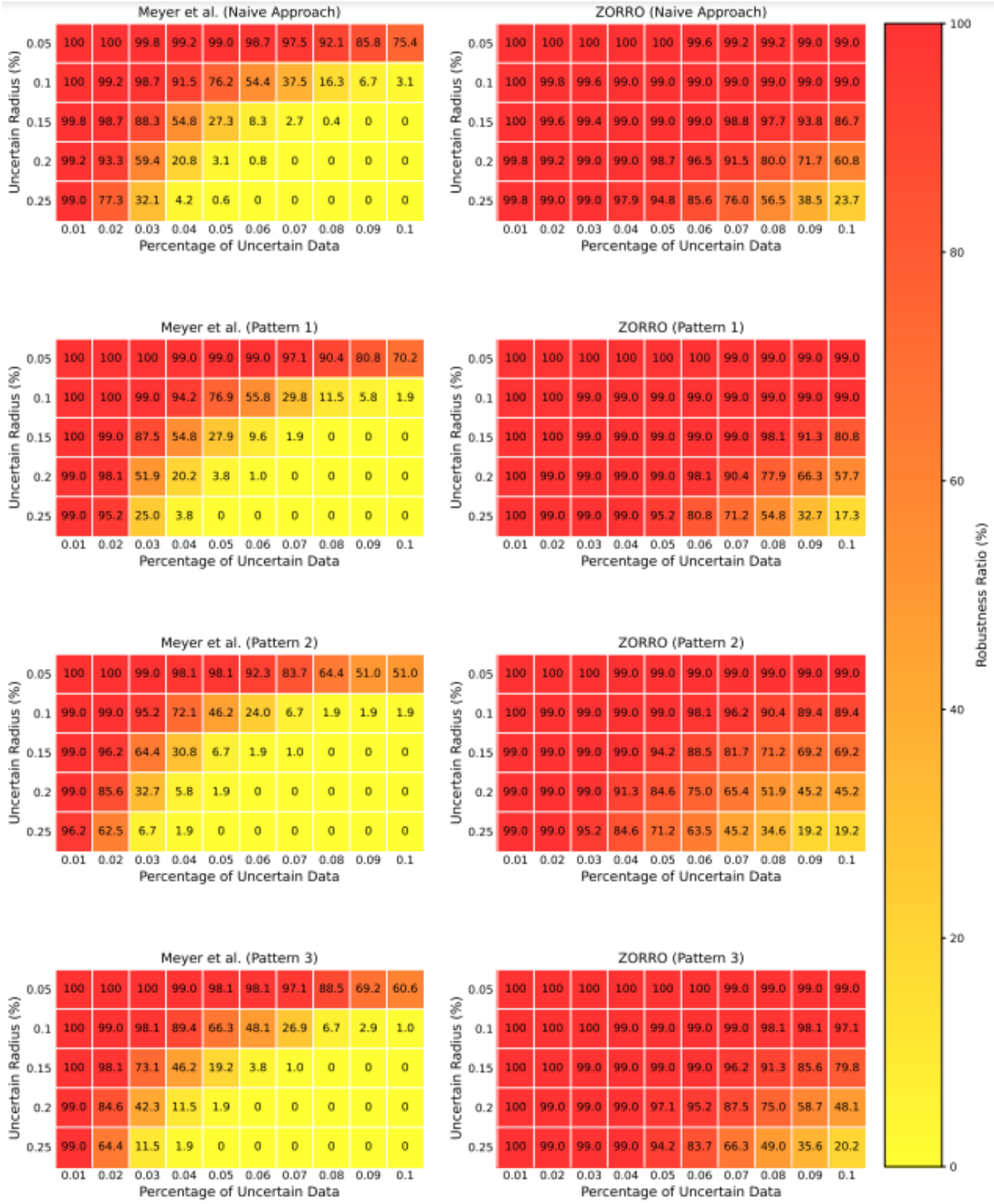

Figure 3: Fire Pattern Mining Heatmap

3.1.4 Boston dataset

The specific best error injection patterns found via pattern mining are as follows:

```
Pattern 1:
'Avg. # of Rooms per Dwelling' > 6.307842105263157
'Pupil-Teacher Ratio per Town' > 17.9

Pattern 2:
'Weighted Mean Distances to 5 Boston Employment Centres' < 2.3514777777777778
'Proportion of Local Population that is Lower Status' < 9.359473684210526

Pattern 3:
'Tax Rate per $10k' > 398.0
'Proportion of Local Population that is Lower Status' < 11.266842105263159
```

Below are our pattern mining results running on the heatmap, represented in figure 4.

On Pattern 1, we reached nearly half the robustness ratio of the naive method, indicating a very successful error injection with pattern mining. Pattern 2 and pattern 3 also remained quite effective too, still outdoing the naive method approach. Thus we can say here that pattern mining was a good method for error injection on our Boston dataset. For reference, the Boston dataset had the lowest sensitivity to labels (tied with MPG), so this follows the idea that pattern mining's effectiveness is based on the sensitivity to the prediction labels of a dataset.

When comparing to the leave one out and histogram approaches, we had a worse case robustness ratio of 11.8 on the leave one out method with a RandomForest utilizing mean squared error, while we had a worse case robustness ratio of 11.8 on the histogram method also with a randomForest utilizing mean squared error. Notice Pattern 2 also achieves a similar robustness ratio of 11.8, but we've also discovered Pattern 1 which achieves an even better ratio of 8.8. So we can safely say in this case the Pattern Mining Approach was the most effective on this dataset here vs all other approaches.

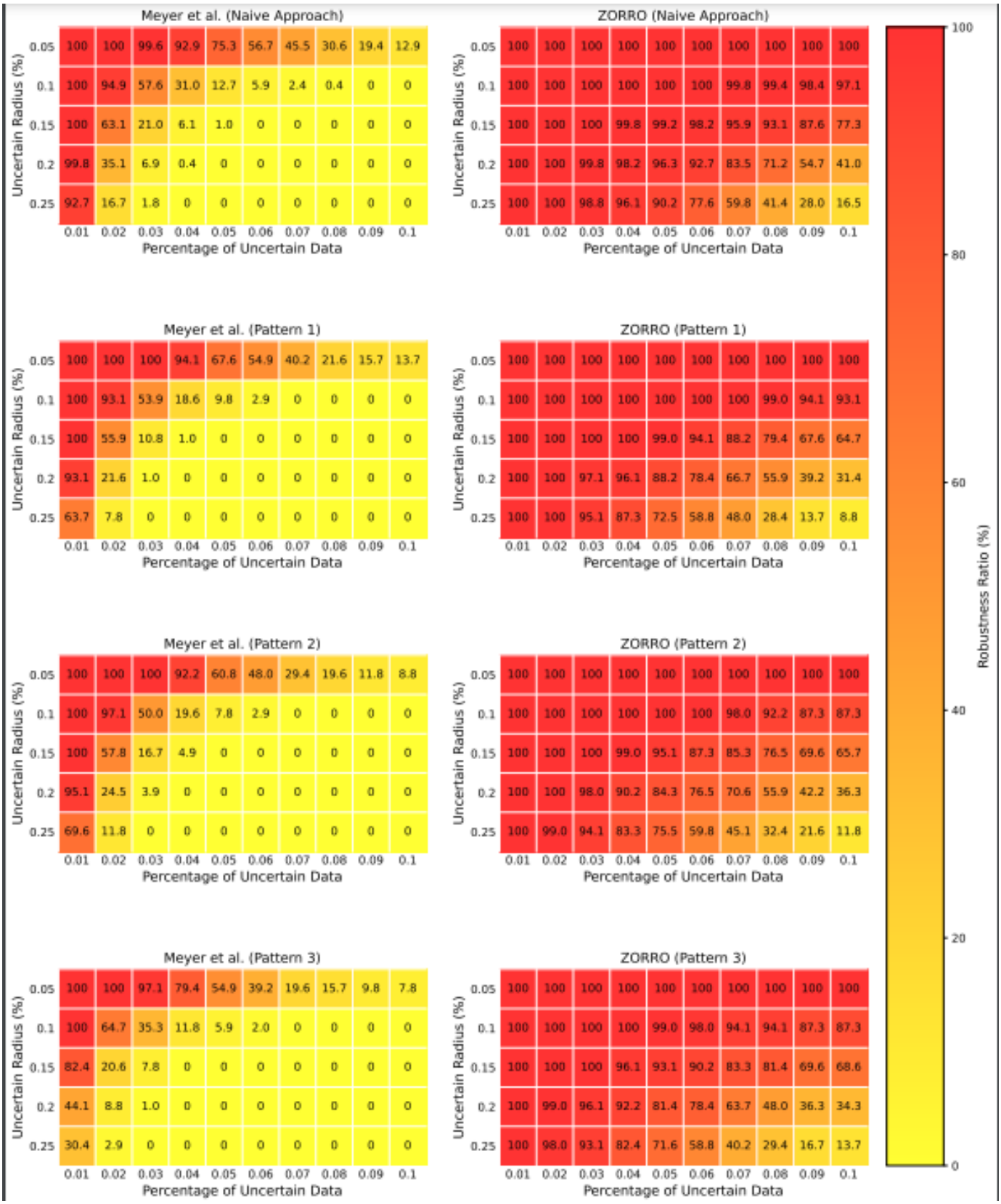

Figure 4: Boston Pattern Mining Heatmap

So overall, we can see that Pattern Mining is quite an effective approach towards our error injection method towards finding the worst case robustness for our dataset, being out the naive random selection approach we had on 3 out of 4 of our datasets. It also seems to be mostly more effective than the leave one out approach and histogram approach in most scenarios, but also reliant on the sensitivity of the dataset (where it was outperformed by leave one out on the somewhat sensitive fire dataset and outperformed by all other approaches on the very sensitive insurance dataset). This is fine however, as it gives a method to determine the worst case robustness with datasets that aren't as sensitive, while more sensitives ones can utilize our previous methods. In the end, pattern mining is quite effective as a error injection tool and we can thus utilize it without our other methods to find a value of measure of robustness on datasets, letting us determine their reliability.

3.2 Pattern Mining Robustness Diminishing

In this section, we'll analyze the patterns of each dataset further under a increasing uncertainty radius for each pattern while keeping uncertainty percentage at 10 percent. This differs in the heatmap in that pattern mining does not order the target indices it finds by importance for error injection, unlike leave one out and the histogram method. Because of that, the heat maps for pattern mining are flawed in that any percentage below 10 percent is a random selection of points from the pattern that is not ordered by importance, hence we don't know for certain this is the worst case scenario. 10 percent differs however under it the patterns will utilize ALL of the indices they capture (since these patterns were mining out of those that capture up to a maximum of 10 percent of the data). Thus by keeping the uncertainty percentage at 10 percent, we can see the diminishment of all the patterns and the respective naive method of random selection points against each dataset to see which method reaches a ratio that signifies the dataset is unreliable or even unusable.

We'll demonstrate each pattern and the naive method as a line plot where x is our uncertainty radius and y is the robustness ratio. As uncertainty increases, the robustness ratio will decrease with it. We'll be looking out for the methods that showcase lines that hit specific threshold y value ratios earlier than the orders. This is because these methods will be signifying that they're decreasing to an unusable level faster than other methods, helping us signify that they're the worst case scenario for that dataset. The specific thresholds we've chosen to analyze on the y axis will be the ratios of 0.8, 0.6, and 0.2. We've chosen these thresholds for specific purposes on each. 0.8 represents the early dip or the head start in the diminishing on robustness we want to analyze. 0.6 is around the area where the dataset would become unreliable. And finally we deem 0.2 to be the point of complete unreliability out of the dataset. We could choose something between 0.5 - 0.2 for the final threshold however. We also do not use 0 as a threshold since by this point the dataset was long already deemed unusable before a robustness ratio may reach such a point.

3.2.1 MPG dataset

Below we showcased the multi-line plot in figure 5 with regards to the 3 patterns found for the MPG dataset alongside the naive method.

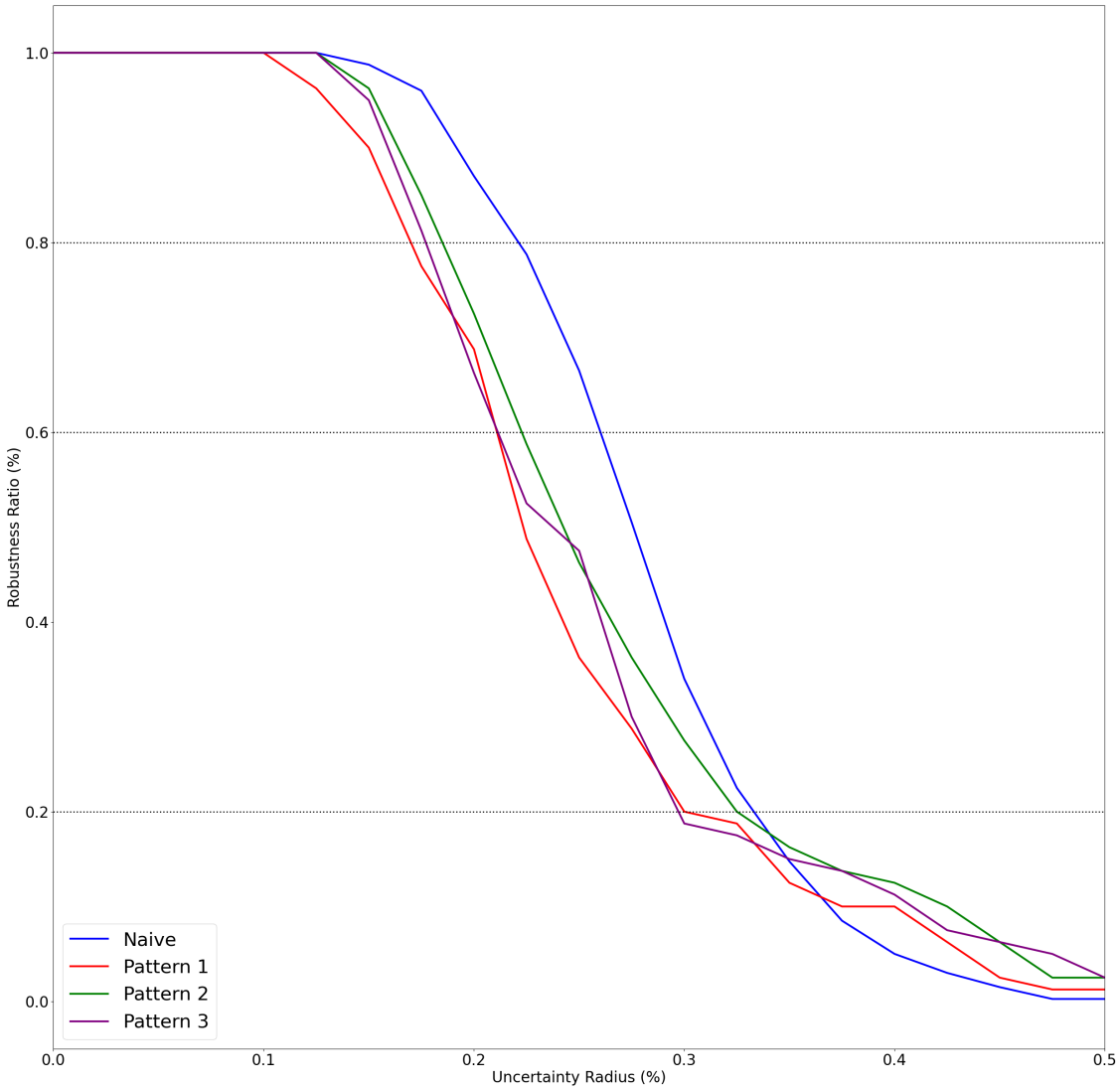

Figure 5: MPG Pattern Mining multi-lineplot

As we can see, there was a lot of instability between pattern 1 and pattern 3 with their zig zag movements across the plot. Meanwhile the Naive method continued a slope format as did pattern 2 (though it started to zig zag near the end). Looking at the dotted lines, we see that there's a pretty wide gap between the naive method and the pattern methods, signifying the larger difference in effectiveness. As for performance, pattern 1 hit the 0.8 ratio first but tied with pattern 3 at the 0.6 ratio. They would continue swapping faster decreasing rates by the 0.6 to 0.2 ratio as well but for most of the run, pattern 1 decreased at a faster rate. Note also by the end that the Naive method completely overtakes all 3

patterns but by then the ratio is already quite unusable, being very close to 0.

3.2.2 INS dataset

Below we showcased the multi-line plot in figure 6 with regards to the 3 patterns found for the INS dataset alongside the naive method.

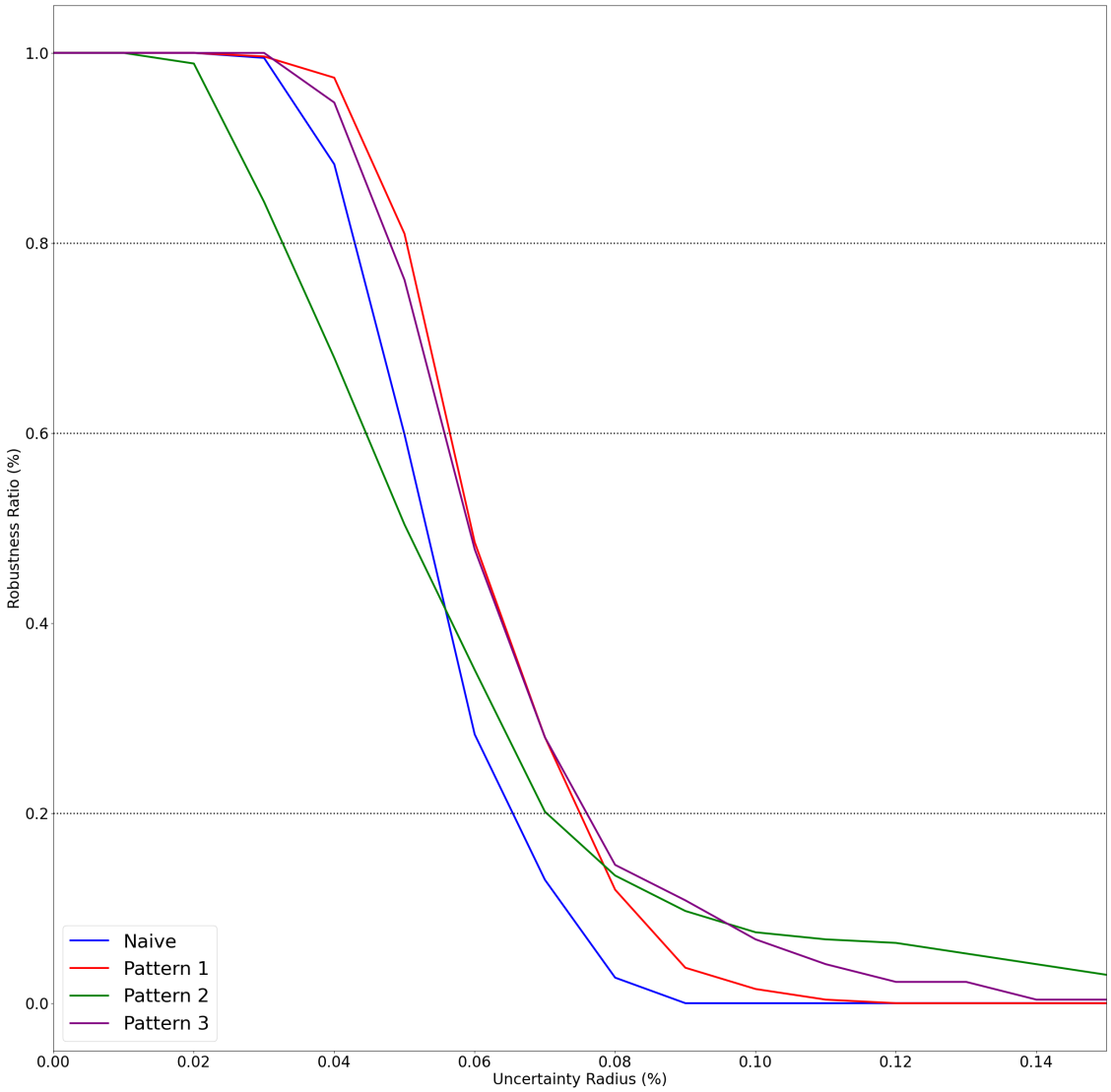

Figure 6: INS Pattern Mining multi-lineplot

First we notice that this time the Naive method isn't as far back to the other line plots as it was in the mpg dataset. We also noticed the plots aren't as aligned with one another as they previously were, Pattern 1, Pattern 2, and the Naive all take varying levels of dip in comparison to the mpg dataset where around the same slope was made but at varying points of uncertainty radius. That said, Pattern 1 and Pattern 3 are somewhat aligned till around a ratio of 0.15.

In terms of robustness diminishment, pattern 2 dominated all 3 patterns till past the 0.08 radius where pattern 1 took focus. However when we factor the naive method on top of it, pattern 2 only lasted till around 0.4 ratio, before it was over taken in the fastness robustness diminishment by the Naive method which remained on top from that point forward. Interestingly, is the rather lack immediate decrease in pattern 2 in comparison to the other methods, especially the gap between pattern 1 and 3. Naive Method would come late too but overtake it with a faster rate in exchange.

The plots here showcase the ability and importance of the sensitivity of the dataset for pattern mining. As we noted, the insurance dataset was labeled to be quite sensitive in comparison to the rest of our datasets and we can see that on full display here. Plots are more spaced out in comparison, and the naive method ended up being similar performance to our patterns here, if not frankly better. That said, there still is some merit for pattern mining here since Pattern 2 could be labeled as a better worse case scenario since by the time ratio reaches around 0.5 (which pattern 2 met before the naive method), a dataset can be deemed ”unreliable” which in certain circumstances would be enough of a reason to not to use it anymore. So while the naive method here reached completely unusability faster, pattern 2 reached the average point of unusability faster on the insurance dataset.

3.2.3 Fire dataset

Below we showcased the multi-line plot in figure 7 with regards to the 3 patterns found for the Fire dataset alongside the naive method.

While the plots carry varying levels of slope here, they all seem to fit a similar shape to one another. That said, the disparity near the beginning before slowly aligning themselves together near the end by around 0.1 ratio is quite interesting, implying there’s a lot more instability between patterns with a smaller uncertainty radius but as it grows larger, all the patterns conform upon one another in a similar manner of robustness diminishment.

First we note that Pattern 2 is largely dominating the early stages of the robustness ratio, overtaking both 0.8 and the 0.6 ratio. It isn’t till around 0.2 that it’s finally overtaken by pattern 1 for slight brief moment before its overtaken by the naive method. Since it’s so dominating for the early and late stages of robustness and by the time it’s over thrown the robustness has already reached a near level of unusability, it can be argued here that the worst case method on the fire dataset would be pattern 2. We’ll be going over a method that can help us confirm this is the case in the next subsection. Past 0.2, the plots are largely non as interesting since even though there is some domination by pattern 1 and the naive method, all the methods are pretty compacted together still by this point.

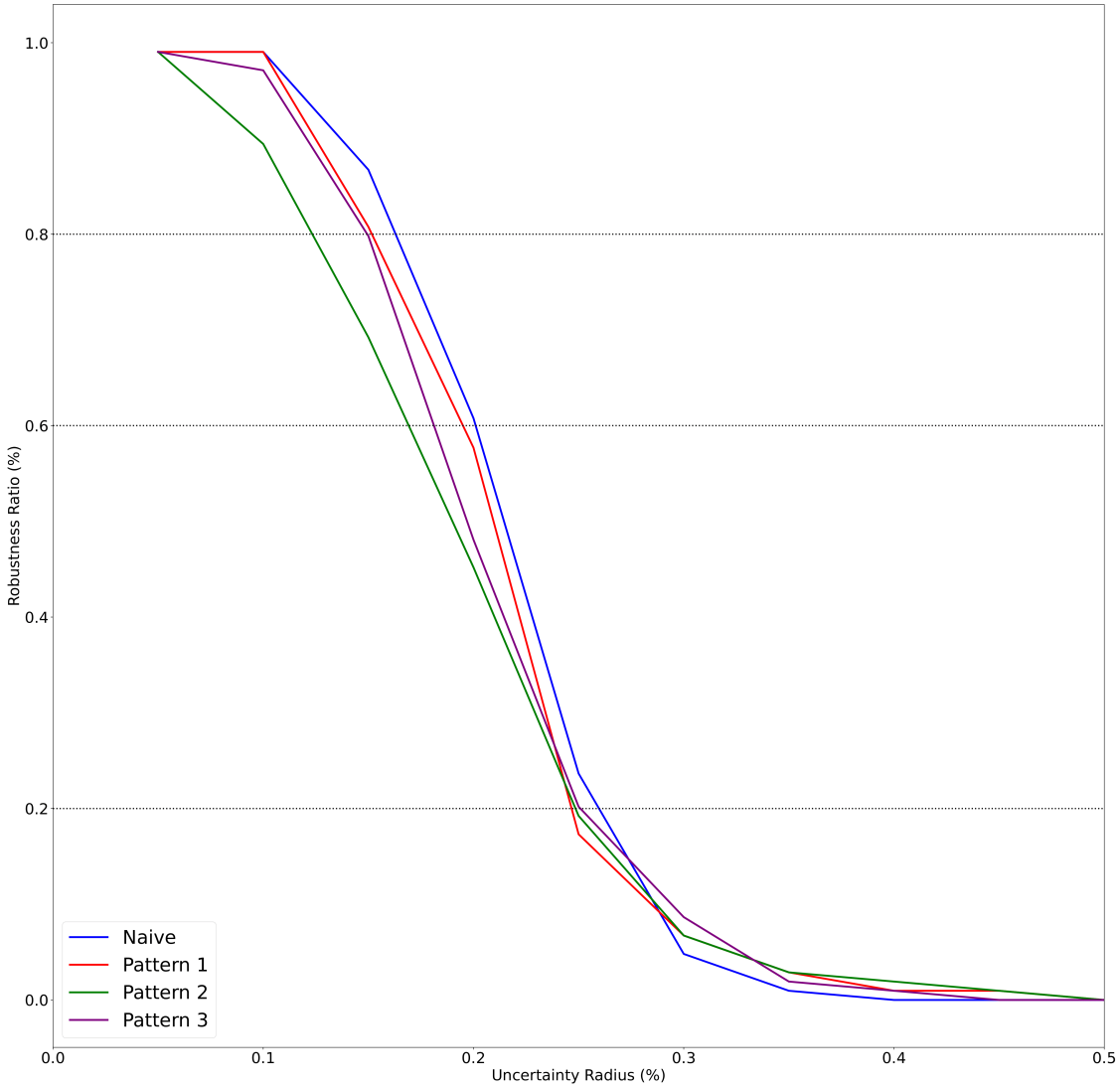

Figure 7: Fire Pattern Mining multi-lineplot

3.2.4 Boston dataset

Below we showcased the multi-line plot in figure 8 with regards to the 3 patterns found for the Boston dataset alongside the naive method.

It seems once again all the plots contained similar sloping to one another, with frankly each method providing a line plot that was matching one another entirely in shape (with the exception of the naive method which dipped a bit below some of the pattern plots). Also noted how crammed together the 3 pattern plots are near the beginning before around the 0.6 robustness ratio pattern 1 starts splitting off from pattern 2 and 3 and at 0.2, pattern 2 starts splitting off from 3 as well. In fact, pattern 2 very closely aligned with pattern 3 for most of the diminishment. There was a brief dip in matching plots near the beginning where pattern 1 didn't immediately drop in robustness as fast as pattern 2 and 3 did. Because of

that, Pattern 2 would be the most decreasing robustness ratio method till around the 0.7 mark. At that point, pattern 1 would take the spot and overshadow it, though not completely as both pattern 2 and 3 still stayed somewhat close to pattern 1 even though it moved away from them (especially close considering how far away and differing the line plots were for the mpg and insurance dataset respectively). After the exchangement of spots around 0.7 between pattern 2 and pattern 1, pattern 1 remained dominating for the remainder of the plot in decreasing robustness ratios till near the end where it sync with pattern 2 and the naive method. Because of that, we get a clear viewpoint that Pattern 1 is indeed the worst case scenario for our fire dataset, since it dominated the rate of decreasing robustness after 0.7 robustness (which could still be deemed usable for dataset standards) and remained the fastest from that point onward, solidifying its position at causing the quickest decrease in robustness to unusability for the fire dataset.

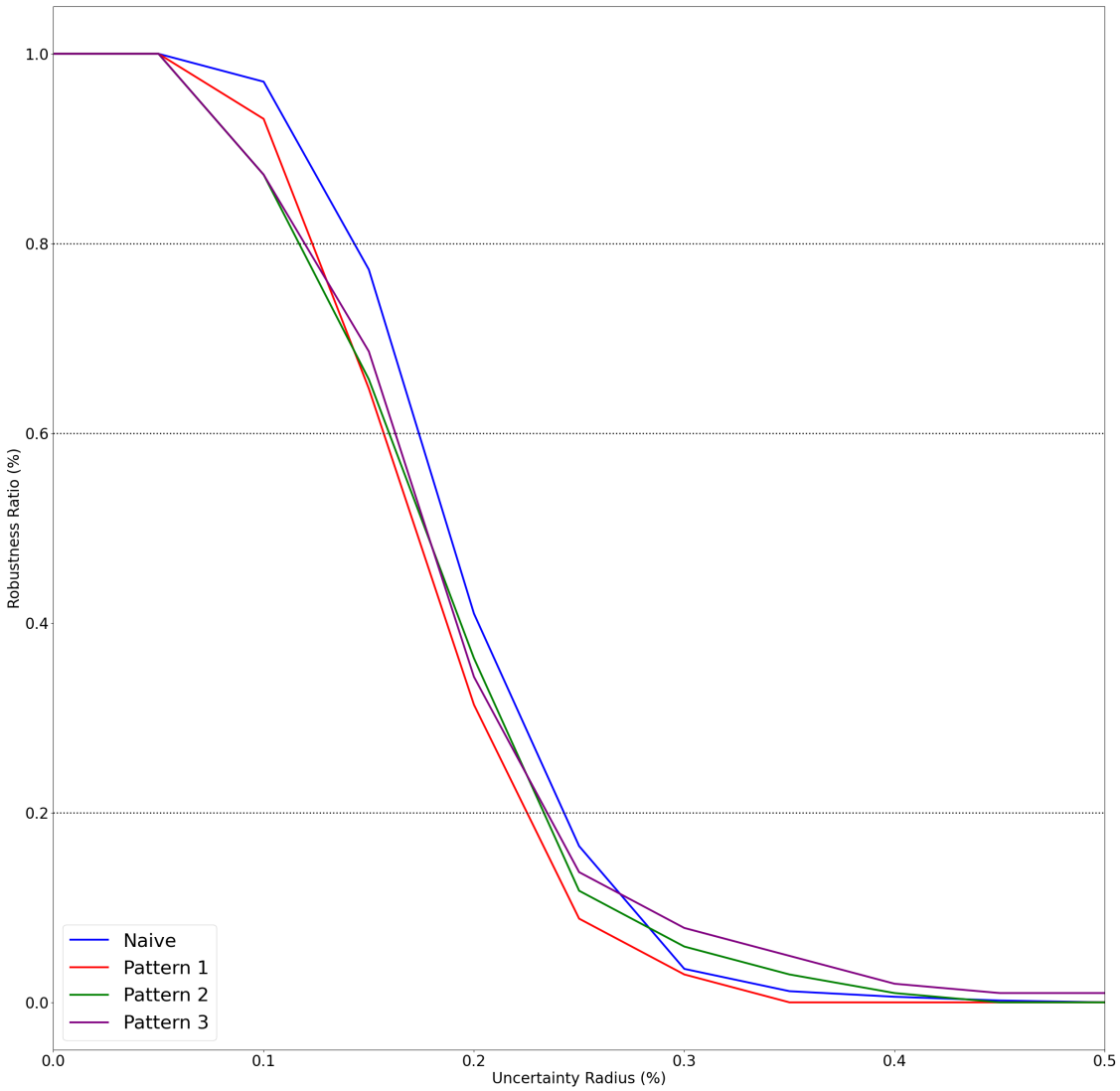

Figure 8: Boston Pattern Mining multi-lineplot

These multi-lineplots help give an idea of how dominating certain methods may be dominating for the diminishment of robustness in comparison to others, along with how effective our pattern mining approach really is. For example, the line plots on the Boston and Fire dataset help highlight the effectiveness of specific pattern injections (pattern 1 and pattern 2 respectively for both dataset) as they remained dominating in reaching lower robustness ratios for most of the run, and also reached points of unusability on said datasets the fastest most of the time vs the other methods they were compared to. That said, other datasets it was harder to pinpoint the more effective methods. For instance, the mpg multi lineplot was quite unstable and even though pattern 1 was dominating the robustness ratio diminishment, it would constantly fight with pattern 3 for the spot so it was hard to conclude if it was truly the worst case scenario. Luckily we have a solution that not only confirms the worst case method but also allows us to compare these worst case scenarios across different datasets to conclude which ones are the most robust with respect to their task labels.

3.3 Normalization

Our plots from previous may showcase some of the effectiveness that pattern mining could bring, but to truly know how effective they are with retrospect to the dataset they're trying to error inject upon, we have to utilize our normalization function which can organize our methods by importance and ability to error inject along with the ability to compare our different datasets' robustness towards one another.

How it works is that we give each a method a set of prerequisite parameters that are the same across each method before we then include boundary threshold we want it to pass (say for example, 0.5 robustness.) Each method is calculated for robustness given these prerequisites and a starting robustness radius. If the calculated ratio fails to pass the threshold, robustness radius is incremented. Once the ratio passes the threshold, the final robustness radius number is recorded. This is done for each method run to collect the respect radii for each method. The radii are then normalized into a scoring tier where the lowest score represents the method that provided the worst case error injection on that dataset.

First we'll showcase the normalization results on our datasets to get a sense of the best error injection method we found for each. Note our below results focused more on the histogram method since running all 3 types of methods proved to induce a large runtime, but the results still remained the same without including the leave one out methods here. (in that pattern mining approaches remained largely dominant)

3.3.1 Mpg dataset

```
Normalized robustness score for MPG_Pattern_Mining(Pattern 1) dataset is -0.4120
Normalized robustness score for MPG_Pattern_Mining(Pattern 3) dataset is 0.1371
Normalized robustness score for MPG_histo(LinReg, MSE) dataset is 0.7549
Normalized robustness score for MPG_Pattern_Mining(Pattern 2) dataset is 1.0294
Normalized robustness score for MPG_histo(Rndfrst, MAE) dataset is 1.1667
Normalized robustness score for MPG_histo(Rndfrst, MSE) dataset is 1.3040
Normalized robustness score for MPG_histo(LinReg, MAE) dataset is 3.0200
```

Figure 9: Normalization Results on MPG dataset

First out of the 4 heuristic histogram patterns seen above in figure 9, the one utilizing the linear regression model with MSE as its metric proved to be the most effective in error injection. We can also get a scale of the effectiveness due to normalization being utilized upon a normal distribution, in that this specific histogram method was twice as effective as the next two methods (the randomforest regressors) and thrice as effective as the last method (linear regression with mae). Adding in our pattern approaches, the only one that ended up worse was pattern 2 with a score of 1.029 vs 0.7549. The other two patterns proved to be much more effective and saw an even lower score, with pattern 3 at 0.1371 and pattern 1 even reaching a negative score at -0.412. The scores are meant to represent the robustness of each method under similar conditions across all of them except the robustness radius (which is the range of value allowed for a given label to be deemed robust). As such the lower the robustness radius, the lower the general robustness on the error injection method, and as such the lower the score, implying the lowest score gives us the best error injection. As such, we can conclude that Pattern 1 for the MPG dataset gives us the best worst case error injection in which we inject error on all indices in the MPG dataset that contain a weight of less than 3376.5, an acceleration of less than 12.12, and a model year of less than 82.

3.3.2 INS dataset

```
Normalized robustness score for INS_Pattern_Mining(Pattern 2) dataset is -0.3767
Normalized robustness score for INS_histo(LinReg, MSE) dataset is -0.0974
Normalized robustness score for INS_histo(Rndfrst, MSE) dataset is 0.4613
Normalized robustness score for INS_histo(Rndfrst, MAE) dataset is 0.8803
Normalized robustness score for INS_histo(LinReg, MAE) dataset is 1.5786
Normalized robustness score for INS_Pattern_Mining(Pattern 1) dataset is 2.2769
Normalized robustness score for INS_Pattern_Mining(Pattern 3) dataset is 2.2769
```

Figure 10: Normalization Results on INS dataset

The normalization for the insurance dataset was quite interesting. Both pattern 1 and pattern 3 were deemed as highly ineffective towards error injection, and all 4 histogram methods have outdone them as well. The main exception is Pattern 2 results which was

concluded by our normalization method as the best error injection for getting the worst case robustness scenario on the insurance dataset. This matches up largely with the multi-line plot provided on the insurance dataset, where Pattern 2 dominated the majority of the robustness range, only losing when the robustness ratio was reduced to near unusability anyway. That said, the spreading result of the pattern method could also be a sign of insurance's sensitivity.

3.3.3 Fire dataset

```
Normalized robustness score for Fire_histo(Rndfrst, MAE) dataset is -0.4366
Normalized robustness score for Fire_Pattern_Mining(Pattern 2) dataset is -0.3594
Normalized robustness score for Fire_Pattern_Mining(Pattern 3) dataset is 0.4636
Normalized robustness score for Fire_Pattern_Mining(Pattern 1) dataset is 1.6981
Normalized robustness score for Fire_histo(LinReg, MAE) dataset is 1.8781
Normalized robustness score for Fire_histo(LinReg, MSE) dataset is 1.8781
Normalized robustness score for Fire_histo(Rndfrst, MSE) dataset is 1.8781
```

Figure 11: Normalization Results on Fire dataset

Both linear regression histogram methods and the randomforest histogram method with a metric for MSE got an equal score for robustness ratio. Meanwhile our pattern methods were deemed more effective for error injection though to varying degrees (with pattern 1 being just barely, pattern 3 to a strong amount, and pattern 2 with the best of the 3 to the point of hitting a negative score.) However, the method of best choice was the histogram method using the randomforest on mean absolute error. This could again be another point towards the idea that the sensitivity of a dataset may give other methods like leave one out and heuristic histogram more leeway vs pattern mining approaches. This is further supported by how we say the pattern scores in the heatmap plots on the fire dataset were not the lowest.

3.3.4 Boston dataset

```
Normalized robustness score for BOS_Pattern_Mining(Pattern 1) dataset is 0.0763
Normalized robustness score for BOS_Pattern_Mining(Pattern 3) dataset is 0.2783
Normalized robustness score for BOS_Pattern_Mining(Pattern 2) dataset is 0.7498
Normalized robustness score for BOS_histo(LinReg, MAE) dataset is 0.8172
Normalized robustness score for BOS_histo(Rndfrst, MSE) dataset is 0.8172
Normalized robustness score for BOS_histo(LinReg, MSE) dataset is 0.9182
Normalized robustness score for BOS_histo(Rndfrst, MAE) dataset is 3.3430
```

Figure 12: Normalization Results on Boston dataset

Finally for the boston dataset, all 3 patterns were deemed more effective than the histogram method with pattern 1 being singled out as the one with the best error injection on the dataset. That said, the gap was rather close across all methods, aside from the histogram

randomforest method using mean absolute error. All other methods tended to be around the similar range, with the largest gaps seen in pattern 3 and 1 which were the best error injection methods.

3.3.5 Comparing against all datasets

Now that all of our datasets have been run on the normalization function, we can highlight their worst case methods and finally run them all against each other! Since we now know the worst case scenario for each dataset, we can then have an idea of the robustness of the datasets against one another! After we gather each worst case method scenario and run the datasets with the normalization function, we get the following results below. Note due to the differing ranges across the datasets' labels, there is some label scaling involved for the normalization comparison to be accurate.

```
Normalized robustness score for Insurance dataset under worst case scenario is -0.2382
Normalized robustness score for FIRE dataset under worst case scenario is 0.2755
Normalized robustness score for BOS dataset under worst case scenario is 1.8923
Normalized robustness score for MPG dataset under worst case scenario is 2.0704
```

Figure 13: Normalization Results on Boston dataset

Overall, it seems that the Insurance dataset was easily the most sensitive of the four datasets we've chosen, being the only robustness score that was in the negatives. Because of this and the rather massive discrepancies between it and the other datasets' scores, we can easily say the insurance dataset was the least robust dataset here and was in fact quite instable in comparison to the other datasets. Next we have the fire dataset who didn't get a negative score but was still on the lower spectrum of a score, being very close to 0. We can't consider it on the same level of robustness as insurance but it's not that far off, and as such can be considered the 3rd least robust dataset of the 4 we chose. Finally we had the Boston and MPG datasets. Both of these scores were on the higher spectrum and were quite cloaw to another(1.89 and 2.07 respectively). MPG barely edges out for the most robust of these datasets, but frankly in our tests, MPG and Boston were quite comparable in robustness to one another. To conclude, we can utilize our normalization function to determine the best error injection method for each dataset which will give us said datasets worst case scenario to utilize upon for comparing all datasets against one another to determine which is the robust given their absolute worst injections. We work in this manner because by finding the error injection that creates the worst robustness ratio for each dataset and comparing said injections to one another, the datasets will all be under worst possible performance and hence when we finally order them, we know said order of robustness will be true since all of them are under the same performance expectation (the worst possible scenario). This in turn allows us to confirm under these circumstances, which datasets are still performing the best (Boston and MPG) and which are much more unreliable for usage (Insurance).

4 Conclusion and Potential Future Studies

The idea behind Zorro was that by finding all possible ranges of data, it could similarly maintain the robustness of a dataset to a much more effective degree than Meyer. Because it works will all possible ranges however, there must exist a group of data such that the worst case injection on Zorro was possible, especially since our original ZORRO method worked by randomization on the target indices. There was also a given parameter that gave a value that represented the range of robustness for that given dataset in ZORRO, which in turn allowed us to represent dataset robustness as a measurement using said range value. With that in mind, we set about finding methods that could find the best error injection target on the dataset (in turning giving us the worst case robustness) which, as expected, incorporated finding methods that discovered the most important indices to predicting the target labels (which would hence make them the best indices to target upon with error injection). Overall, we found 3 different methods in leave one out, histogram, and pattern mining; each method having varying levels of success at this task. Pattern mining in the end represented the best overall error injection, having the lowest error injections on average but was less usable on more unstable range datasets such as the insurance dataset which would better utilize leave one out approaches due to said instability. Meanwhile the histogram approach would be largely outclassed by the two methods and more so in the middle for error injection wise as a jack of all trades, master of none. After gathering all worst case methods for error injection on each dataset utilizing our normalization function which depended on the aforementioned parameter that represented range of robustness, we then gathered all our datasets and were able to compare them once more for an idea of how robust each dataset was with respect to one another

With this study, we were able to accomplish two things, 1) we were able to create error injection methods on datasets with ZORRO that could tell us the most important indices to watch out for instability upon prediction and 2) we were able to compare usability across datasets in terms of robustness allowing us to choose which dataset to utilize in a given task. By accomplishing these two things, we can gain better understanding in our datasets and have a larger confidence on the utilization and reliability of datasets which in term would lead to larger performance increases and dirty dataset usage.

The next steps in our study should focus on the simplification of our robustness values across these sets. This involves better addressing the scaling across the robustness values and exploring the numerical differences across datasets and methods. While we can infer the differences across the normal distribution, getting a sense across an infinite possibilities of robustness values could give a concrete answer towards how much better or more robust one method/dataset can be towards another. We can also further explore pattern mining, as it can be compounded upon further and new filter methods can be created to find better patterns or simply find the same ”best” patterns we found but faster.

5 Appendix

A dataset is important in understanding a given group's research and statistical information, as it can uncover insights and pattern trends in the data that are useful for future decision-making. However, the dataset we use may not always be truly representative of the actual ground truth of the domain, largely due to inconsistencies such as outliers, missing values, and errors in the data. This raises the question of how we can claim a dataset is robust, i.e., representative of the true data despite these inconsistencies.

To address this, we utilize an error injection method to test the robustness of a given train-test dataset. This is simpler than it sounds: we inject variance into numbers, reducing their correlation with other data features. The larger issue lies in satisfying the conditions required for effective robustness testing. Our research found, unsurprisingly, that effective robustness tests and corresponding error injections are directly correlated with distributions that are close to representing the true distribution of a given dataset.

Current methods for identifying this distribution rely on interval parameters or large-scale distribution testing. Interval parameters result in larger robustness ranges, which is unsatisfactory for addressing uncertainty in data. On the other hand, large-scale distribution testing is highly effective in maintaining robustness despite error injection but is computationally expensive, as it learns from all possible distributions in the data, leading to a very large time complexity, $O(n^n)$

To mitigate this, we consider heuristic data mining approaches inspired by systems like Gopher. These methods approximate close-to-true distributions without incurring the prohibitive computational costs of exhaustive learning. This approach enables the creation of robust classifiers that maintain efficiency while avoiding the complexity of learning every possible distribution. By applying this setup during error injection, we can measure success by comparing the effectiveness of heuristic methods against traditional large-scale testing, ensuring lower computational cost without sacrificing robustness. This can be seen in our pattern mining approaches where we were able to discover patterns that provided very effective error injections on the dataset while serving as the worst case scenarios for said dataset as well, allowing us to get a strong understanding of the possible robustness of a dataset. Ultimately, this enables faster and more reliable evaluation of a dataset's robustness by assessing its performance under worst-case scenarios, measuring the ratio within a robustness range relative to the true distribution. In conjunction with the normalization of robustness values across datasets, allows us to provide evidence that certain patterns are the most effective at robustness error along with the idea that datasets can be compared for robustness with said normalization function

6 Contributions

- Joshua Huang: Mainly led code optimization (including both creation and setup along with actual test runs) along with code replication confirmation (both setting things up and moving code from testing to terminal based usage), helped to write

report along with the data gathering and cleaning

- Marlon Garay: Helped out with code testing and report writing, created and edited writing references such as the proposal, ethics and schedule. The code and direction were mostly done with help under guidance of Joshua. The ideas, visuals, direction, and design of the website was under the guidance of Joshua and Marlon.

References

Gopher. 2024. "Gopher System: Robustness and Explanation Papers." https://gopher-sys.github.io/index.html

Marlon Garay, Joshua Huang. 2024. "Using Error Injection to Determine the Robustness of a Given Dataset." https://docs.google.com/document/d/1FoiUCxiMsFaBJMHtvNRZXgJ4mQVd7XqtAp5Wr--bVe4/edit?usp=sharing

Zhu, Jiongli, Su Feng, Boris Glavic, and Babak Salimi. 2024. "Learning from Uncertain Data: From Possible Worlds to Possible Models." [Link]