|  |  |  |
| --- | --- | --- |
| **Fall 2024** | **Report #2 – 09/10/2024** | **Dingyi Nie** |

**——————————————————————————————————————**

**I. Task Achieved Last Week**

**——————————————————————————————————————**

* + Examined closely how XGBoost handles missing value (NaNs);
  + Cleaned PhysioNet Challenge 2012 dataset and aligned the preparation and split with previous related works (GRU-D, RAINDROP and ViTST).

**——————————————————————————————————————**

**II. Feedback and Interaction**

**——————————————————————————————————————**

* **Prof. Kuo’s Feedback**
  + Last week we comprehensively examined XGBoost’s discriminant abilities on PhysioNet Challenge 2019 (P19) data, without any feature extraction, selection and generation. We discovered that on the Sepsis occurrence binary classification task, XGBoost performs the best when no imputation is applied (i.e. missing values in the input are NaNs). All 7 imputation strategies only degrade its classification AUROC. This can be explained with the viewpoint proposed by GRU-D, that in many medical datasets the missingness and temporal missing patterns and actually meaningful, and XGBoost alone can make use of this information. We need to see how XGBoost is handling missing values internally.

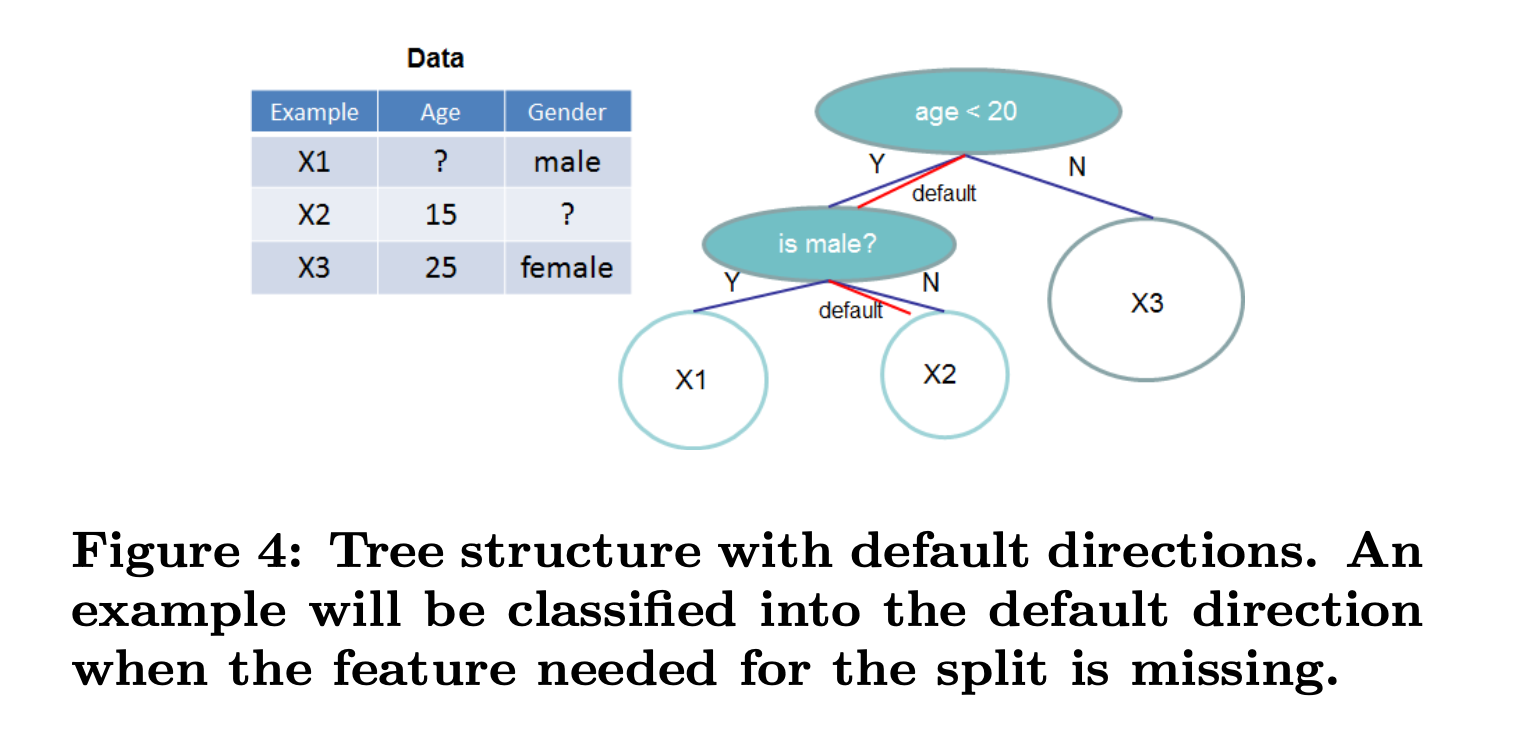
**——————————————————————————————————————**

**III. Report**

**——————————————————————————————————————**

**How XGBoost handles input missingness**

XGBoost has an inherent module that deals with missing values called sparsity-aware split finding. In a nutshell, default branch directions for missing values (NaNs) are learned during training in its tree algorithms. The selection of default branch directions minimizes the loss.



This can introduce extra non-linearity associated with missing features. It also explains the difference in the model’s AUROC score metric before and after imputation, since after imputation all NaNs become real values and are treated no differently than regular observed values, resulting in decreased model capability.

I have conducted 2 simple experiments to verify that XGBoost can actually capture the correlation between missing rates and potentially the temporal pattern of missing rates.

**Experiment 1**: Generate a synthetic dataset consisting of 1000 data samples. Each sample is a 100 by 10 matrix. Consider 100 to be the sequence length dimension (time axis) while 10 being the number of features. Each sample is associated with a binary classification label 0 and 1. Half of the samples belongs to class 0 while the other half belongs to class 1. To fill-in the values, we sample from a Gaussian distribution *N*(0, 1). Then, we manually assign random NaNs to all the data samples. For data samples that belong to class 0, all feature columns have a missing rate of 0.1 (i.e., every value in the same column has a chance of 10% to become NaN); for class 1 samples however, while every other column has a 0.1 missing rate, the last column will have a 0.2 missing rate. Hence, the missing rate of the last feature is correlated to the data label in our synthetic dataset. We divide the dataset to 4:1 train-test ratio, fit an XGBoost to the raw train data and test the AUROC on test set. We also do 3 types of imputations: zero-imputer (replace all NaNs with 0’s), mean-imputer (replace all NaNs with average feature value on train set) and linear-imputer (need to specifically handle columns with only 1 observation or no observation), and then test the imputed data and compare with the raw data. Table 1 shows the results.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| AUROC | Raw | Zero-imputed | Mean-imputed | Linear-imputed |
| Exp. 1 | 78.48% | 63.97% | 63.97% | 61.43% |
| Exp. 1 | 77.49% | 63.34% | 60.76% | 59.22% |
| Exp. 3 | 84.69% | 63.21% | 67.56% | 66.55% |
| Exp. 4 | 79.68% | 60.35% | 54.84% | 54.07% |
| Exp. 5 | 84.56% | 59.29% | 60.42% | 61.19% |
| **Avg** | **80.98%** | **62.03%** | **61.51%** | **60.49%** |

Table 1. Experiment 1 – missing rate correlations.

**Experiment 2**: Similar settings to the previous experiment. The dataset size, split and sample shape are the same. Class 0’s data characteristics (Gaussian parameters, missing rates) are the same as well. The only difference is for the last column of class 1’s data – instead of having a higher missing rate of 0.2, it has an oscillating missing probability that goes up and down with respect to the time axis: p(xt=NaN | t) = 0.1 + 0.05 sin(πt), where t = {0, 1, …, 99}. The overall missing rate is still 0.1, but it just has a special temporal pattern of missingness.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| AUROC | Raw | Zero-imputed | Mean-imputed | Linear-imputed |
| Exp. 1 | 56.59% | 54.95% | 51.00% | 48.27% |
| Exp. 1 | 55.90% | 44.56% | 46.78% | 47.56% |
| Exp. 3 | 52.96% | 49.72% | 46.75% | 49.28% |
| Exp. 4 | 62.06% | 59.12% | 51.91% | 51.12% |
| Exp. 5 | 49.07% | 48.06% | 53.90% | 47.91% |
| **Avg** | **55.31%** | **51.28%** | **50.07%** | **48.83%** |

Table 2. Experiment 2 – temporal missing patterns.

These two experiments clearly show that XGBoost can benefit from input sparse data with missing values when missingness in the dataset is somewhat meaningful. Here we recall the results from previous week about a single XGBoost classifier’s performance on the real-world P19 dataset and its imputed versions:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Imputer | None | Zero | Mean | Forward | Linear | Spline | Kmeans | XGBoost |
| AUROC | **95.26%** | **94.69%** | **94.60%** | 93.78% | **93.18%** | 92.86% | 94.18% | 91.51% |

Table 3. XGBoost on P19 and imputed P19’s.

Note how similar the pattern is regarding the decreasing tendency in AUROC metric from no imputation to zero-imputer, to mean-imputer, and finally, to linear-imputer. We can consider missing values as “outliers”, that are easily distinguishable from real-observed data. If an imputer is so good that it can perfectly smooth out the data, it will make the missing values no longer discernable and mix into real data. While this can be desired sometimes, especially if the downstream model require the input to be well-defined, dense matrices, it loses all the discriminant features that may be exploited from the missing patterns.

To draw a conclusion for this section, we can say that by adding a dedicated module that explicitly handles missing values in a sparse input, we are likely to see a performance gain. However, feature extraction tools in GL such as Saab transform expect well defined, dense input. A combination (concatenation) of imputed full time series and an explicit representation of the missing pattern (e.g. a mask matrix) might be helpful.

**Preparations for a new dataset: P12**

**——————————————————————————————————————**

**IV. Next Steps**

——————————————————————————————————————

* Run previous GL pipeline with P12 dataset and analyze the results.
* Compare performance when missing data is explicitly provided and not.

**——————————————————————————————————————**

**V. Milestone**

——————————————————————————————————————

* Had an clear insight on providing missing patterns explicitly as input.
* Codes are available here: <https://github.com/d9sus4/GL-TS>