Agenda:

* Iterative thresholding to remove data
* Run different machine learning models that can handle missing data

**Iterative Thresholding**

I iteratively removed columns and rows from the dataset that contained more NAN values than the defined threshold. As the threshold increased, the proportion of NAN values left in the dataset also increased. This occurred regardless of whether columns or rows were removed first. However, when columns were removed first, the dataset sometimes had a higher proportion of NAN values remaining than when rows were removed first for the same threshold. A threshold of approximately 65% and below produced a dataset containing less than 20% of NAN values.

A graph with numbers and dots

AI-generated content may be incorrect.

Removing rows then columns versus columns then rows produced a similar number of columns/features (~15 to 20) and rows/samples (~5200 to 6800) remaining in the dataset. For thresholds of 70% and 90%, there were larger differences in the number of rows left depending on the order of thresholding (by 800 to 1000). The number of rows left in the dataset increased by approximately 800 when the threshold increased from 60 to 65%.

A graph with numbers and lines

AI-generated content may be incorrect.A graph with numbers and dots

AI-generated content may be incorrect.

I conducted a small experiment where I applied a third layer of thresholding. When I removed columns then rows with more than 90% NAN values, then subsequently removed columns with more than 70% NAN values, 29% of my dataset consisted of NAN values. The same proportion of NAN values remained when I initially thresholded by 80%. This is a similar proportion of NAN values left in the dataset compared to only 2 layers of 70% thresholding. It also produced datasets with a very similar number of rows and columns than when I only thresholded with 2 layers (15 features and 6790 and 6771 rows depending on 90 or 80% thresholding respectively).

**ML Models that can Handle Missing Data** (using data from 2000 to 2024)

Goal:

To predict the Maternal Mortality Ratio using a range of features from the WHO and World Bank datasets.

I tested a range of machine learning models that can handle missing data, as well as models that cannot for which I have used imputed data. I conducted this testing for training/testing splits done **by** year and **within** year.

I quantified each model’s error using the mean absolute error, which is the average of the absolute differences between the model’s predictions and the ground truth label for each input. By not squaring the difference, I do not overestimate the impact of outliers.

The units of the errors are number of maternal deaths per 100,000 live births.

*Models used:*

XGBoost (Extreme Gradient Boosting)

Combines the output of multiple models so each new model works on reducing the error (residuals) of the previous combined models.

<https://xgboost.readthedocs.io/en/stable/tutorials/model.html>

LightGBM

Uses gradient boosting but with optimisations that allow the model to run quickly. For instance:

1. Exclusive feature bundling
   1. Creates a new feature that takes one value if variable 1 is true and another if variable 2 is true (because they are exclusively true)
2. Gradient based 1 side sampling
   1. Takes a proportion of data points with the largest gradients (capturing the most error) and a subset of smallest gradients (model doing the best)
      1. Keep all datapoints with large gradients and only a sample of the small gradients (less informative)

Decision Tree

<https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html#sklearn.tree.DecisionTreeRegressor>

Predicts a value through producing a tree with binary decision nodes. The nodes are created to maximise entropy gain. However, decision trees are highly sensitive to training data and may be difficult to generalise.

Random Forest (Bootstrapping + aggregation = bagging)

Random forest uses many decision trees. Creates new datasets from original by random sampling with replacement (same number of rows as original), but with each dataset only containing data for randomly selected subset of features. Then design decision trees for each of these datasets. You can pass new datapoints through all the decision trees and use majority voting to determine the aggregate prediction.

*Imputation methods:*

KNN Nearest Neighbours Full Imputation

Uses the closest values to the missing data point for imputation. This method can have low precision and can introduce non-existent relationships between variables (<https://pmc.ncbi.nlm.nih.gov/articles/PMC8549433/#Sec24>).

Miss Forest Full Imputation

This method initially imputes data using a naïve method (e.g. using the mean), then for each feature with missing values, it fits and trains a random forest on the existing values to be able to predict these missing values.

Polynomial Interpolation

Using the Pandas interpolation method to impute missing data within the existing data range by fitting an order 2 polynomial to the feature’s data.

Iterative Data Winnowing

I removed columns and then rows that had more than 80% NAN values and fed the smaller dataset to the model. This threshold produced a dataset with approximately 40% NAN values, 6771 rows/samples, and 19 columns/features.

*How the methods were combined:*

Only XGBoost and LightGBM could work with data that contained missing values. I tested these models with no imputation, thresholding and polynomial interpolation.

In contrast, the decision tree and random forest models could not work with missing data. Before fitting and training these models, I imputed the data using all the different imputation methods discussed above.

* **Results when splitting data within years**

I used Scipy’s train/test function to split the unprocessed data into 75%:25% training/testing sets. I stratified the data by ‘date’ to ensure that there were the same proportions of the different dates in both the training and testing sets.

A white sheet with blue dots

AI-generated content may be incorrect.

Observations:

* Random forests performed better than decision trees.
* KNN and Miss Forest performed better when there was 80% thresholding.
* Polynomial imputation performed better than thresholding for LightGBM and XGBoost.
* XCGoost with polynomial interpolation performed the best of any of the models, produced mean absolute error of ~15.5 maternal deaths per 100,000 live births.
  + However, this was very similar to the second-best model (LGBM with polynomial imputation, MAE of ~16.6 maternal deaths per 100,000 live births).
* **Results when splitting data by year**

I defined the training data to consist of all data collected between 2000 and 2018. The testing data was defined as all data collected after 2018.

A screenshot of a computer

AI-generated content may be incorrect.

* Observations:
  + Random forests performed better than decision trees.
  + 80% thresholding for KNNs and Miss Forest performed the best.
  + XGBoost and LGBM with polynomial imputation had the best model performance, with their mean average error being ~21.7 and 23.6 maternal deaths per 100,000 live births, respectively.
* **Split by year versus split within year**

**Split by Year**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Split within Year**

**A white rectangular object with black text

AI-generated content may be incorrect.**

Observations:

* Regardless of the type of split, XGBoost with polynomial imputation performed the best of any model. However, the worst-best model when splitting within year outperformed the best model when splitting by year (MAE ~20.9 versus 21.7).
* The best models differed by approximately 6 maternal deaths per 100,000 live births (15.5 versus 21.7 MAE).

To-Do:

* Fine tune hyperparameters (Optuna)