Agenda:

* Explanation behind dropped columns
* Fine-tuning models with 95% datasets
* Fine-tuning models with 100% datasets

**Dropped Columns**

I dropped the following columns from the input dataset

* Lifetime risk of maternal death (1 in: rate varies by country)
* Lifetime risk of maternal death (%)
* Number of maternal deaths

The ‘number of maternal deaths’ is an annual estimate of the number of women who die while pregnant or within 42 days of termination of pregnancy, irrespective of the duration and site of pregnancy, from any cause related to or aggravated by the pregnancy or its management but not from accidental or incidental causes.

* This is basically the same definition as the maternal mortality ratio, and was thus excluded to prevent the model from ‘cheating’

The variable ‘Lifetime risk of maternal death (%)’ had a Pearson’s correlation coefficient of ~0.97 with the modelled MMR estimate. Thus, I dropped it from the table.

* ‘Life time risk of maternal death is the probability that a 15-year-old female will die eventually from a maternal cause assuming that current levels of fertility and mortality (including maternal mortality) do not change in the future, taking into account competing causes of death.’
* I dropped the other lifetime risk because it is a very similar measure to the other lifetime risk.

**Fine-tuning models on the 95% dataset**

Caveat: I am using the fine-tuning results for the RNN and MLP from when I dropped the three columns listed above and two additional columns (which were not dropped after correlation analysis) because my kernel crashed every time I tried to use the Optuna software with these structures. I am unsure of why this is happening and eventually had to move on for purposes of making progress.

I fine-tuned models based on correlation and non-correlation imputed data (not standardised data, as generally the models performed more badly with standardisation and so this was not judged to be a good use of time).

I ran 300 trials over each type of dataset and model choice for all models except for Scikit Learn’s Random Forest, for which I ran 30 trials (would still take ~4 hours).

Scikit Learn’s Random Forest:

* n\_trees = trial.suggest\_int("number\_trees", 10, 200)
* bootstrapping = trial.suggest\_categorical("bootstrapping", [True, False])
  + Generally, bootstrapping was used by best model (with two exceptions)
* max\_depth = trial.suggest\_int("max\_tree\_depth", 5, 40)
* min\_sample\_split = trial.suggest\_int("min\_sample\_split", 2, 10)

Generally, the best model did not approach the upper bounds of the number of trees or max depth.

LightGBM:

* n\_trees = trial.suggest\_int("number\_trees", 10, 200)
* max\_depth = trial.suggest\_int("max\_tree\_depth", 5, 40)
* boosting\_type = trial.suggest\_categorical("boosting\_type", ['gbdt', 'dart'])
  + Dart boosting randomly drops out trees, analogous to drop out in neural network, to reduce overfitting
* learning\_rate = trial.suggest\_float("learning\_rate", 0.0001, 0.1)
* l1\_norm = trial.suggest\_float("l1\_norm", 0, 0.001)
* l2\_norm = trial.suggest\_float("l2\_norm", 0, 0.001)

XGBoost:

* n\_trees = trial.suggest\_int("number\_trees", 10, 200)
* max\_depth = trial.suggest\_int("max\_tree\_depth", 5, 40)
* boosting\_type = trial.suggest\_categorical("boosting\_type", ['gbtree', 'dart'])
* learning\_rate = trial.suggest\_float("learning\_rate", 0, 1)
* l1\_norm = trial.suggest\_float("l1\_norm", 0, 0.001)
* l2\_norm = trial.suggest\_float("l2\_norm", 0, 0.001)

Scikit Learn’s AdaBoost:

* n\_trees = trial.suggest\_int("number\_trees", 10, 200)
* loss\_type = trial.suggest\_categorical("loss\_type", ['linear', 'square', 'exponential'])
  + #loss function used to update weights after each boosting iteration
* learning\_rate = trial.suggest\_float("learning\_rate", 0, 1)
  + #scales contribution of each regressor at each boosting iteratio

Scikit Learn’s SVM:

I reduced the range of the hyperparameters tuned for this model because wider range of possible hyperparameters caused the process to pause during the training process as the model struggled to converge.

* degree = trial.suggest\_int("\_polydegree", 2, 10)
  + I increased degree to 10 because I was consistently having the best models be on the upper end of the range. However, despite the best models tending to have higher degree polynomials (with 3 having degree of 10), I did not further increase the polynomial degree to prevent the model from overfitting.
* kernel = trial.suggest\_categorical("kernel", ['rbf', 'poly'])
  + I did not use the ‘linear’ or ‘sigmoid’ kernels because these seemed to hurt the model’s ability to converge.
* epsilon = trial.suggest\_float("epsilon", 0.05, 0.5)
* regulariser = trial.suggest\_float("regulariser", 0.5, 1)

Scikit Learn’s KNN Regressor:

* n\_neighbours = trial.suggest\_int("number\_neighbours", 2, 85)
  + #read a source that suggested best number of neighbours can be square root of number of samples
* weights = trial.suggest\_categorical("weights", ['uniform', 'distance'])
  + #weights neighbours equally or by distance
* p\_num = trial.suggest\_int("p", 1, 2)
  + 1 = manhatten distance
  + 2 = Euclidean distance

Knn: sometimes suggested to be square root of samples

<https://pmc.ncbi.nlm.nih.gov/articles/PMC4916348/#r6>

More trees better, but largest performance gain in first 100

<https://jmlr.org/papers/volume18/17-269/17-269.pdf>

Visualised results:

A graph with dots and numbers

AI-generated content may be incorrect.

A screen shot of a graph

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.

A screen shot of a graph

AI-generated content may be incorrect.

A screen shot of a graph

AI-generated content may be incorrect.

Observations:

* KNN and Miss Forest imputation appeared to have similar relative model performance, with the KNN regressor model consistently performing the best.
* The polynomial imputation methods also appeared to show the same general pattern, but a different pattern than that exhibited by the KNN and Miss Forest methods.
  + Neural networks and LightGBM or XGBoost tended to perform the best.
* With one exception, all models trained with Miss Forest data did best when correlation imputation was applied.
* There was more variation when trained with KNN imputed data, but the best performing models did not have correlation imputation
* A bit of variation with the polynomial imputation datasets
  + The neural networks tended to do best without correlation imputation
  + The ensemble based methods tended to do better with correlation imputation (except AdaBoost, which varied)
  + KNN regressor did better with correlation imputation but SVM varied.

**Split by countries:**

* Within high-income: Divide data into 80% training, 10% validation, 10% testing, equally between four income-level
* Within low-income: same
* With medium-income

-> Concatenate into 80, 10, 10

\* Training different models on 80%, use the 10% validation as stopping rule of training.

\* Testing all models on 10% test set, extract

- impute data either with all 5 methods and average result or choose totally new imputation method (test 100% on 95% testing sets)

Splits by years

The same country can only be in one of the sets to keep this independent

* By year is already independent