**Boosted Trees**

XGBoost is a flexible model for predicting patient admission given sets of data of both the numeric kind and the categorical kind. The basis of XGBoost is a decision tree. In a decision tree, a group of data points is split into multiple groups with respect to a certain feature, for example, splitting a population into age groups. Multiple decision trees can be made for a group of data. For each tree, one can assign a prediction score for each split and combining all prediction scores over all trees yield a prediction score for each data point.

For each data point, the score is given by . We train the tree to minimize a metric based on each with the form

where is the loss function and is the complexity of the tree. The training of the tree involves learning the functions . The algorithm does so additively, i.e., given for

the additional is chosen to maximize the at the next step. The loss function is chosen to be a quadratic approximation of some function, such as the mean square error and the complexity function is a prescribed form. The optimal can be derived from the closed form objective function.

The next step is to learn the tree structure. This cannot be done by generating all trees and selecting the best one. The optimization will happen one level of the tree at a time by calculating the gain obtained at each node if it further branches into two other nodes.

**XGBoost**

Suppose a naïve model is used to predict the class of some rows. As it is a naïve model, it will have errors. These errors require an update to the model by using a regression tree to approximate the error. Each time a new regression tree is used to update the model, the loss function is being progressively minimized through gradient descent.

Mathematically, let the naïve model be such that for points ,

To improve this model, let be a regression tree such that . Then actually, the tree , so it approximates the error. If the loss function is defined as the MSE,

then approximates the gradient of the loss function. Hence we can update with and repeat the process.

**Code problems**

1. Design matrix and prediction matrix is now room-specific. If one were to create a general model, where the room is an indicator, it appears to mess up the xgboost package.
2. The inclusion of the flow reading epi\_infusion also messes with the xgboost package.
3. Rpart is a package that contains a setting to show the importance of an indicator. However, the library that runs rpart will interfere with the package that runs the original xgboost model. I have separated the xgboost prediction on one file (xgboost\_model) and the rpart on another (xgboost\_model\_rpart).

**Outlines of further extensions**

1. Trajectory information. We can add as a feature the trajectory each patient took at their current stage. So someone may have done triage – resus – majors. At the moment we separate this patient out to have three rows, but we can perhaps add in the majors row the trajectory triage – resus. This should be done when compiling bed moves and most likely have a glossary of shortened names for convenience. You probably know the most efficient way; I would group rows by csn and then order them in admission time, then progressively chain the rooms together. I would also suggest adding this trajectory in a new column and leave the room column as is, so later it may be convenient to take the trajectory off consideration if needed.
2. Length of stay. We can add as a feature the amount of time someone has been in ED. There are many ways to go about it. Perhaps we can set different duration thresholds such as >1, >2, >3 hours. Then we can add a new column of duration for each patient. This also means that if a patient stays at one stage for over one hour, it may be needed for the patient at that stage to be duplicated with newer

**Readings**

1. <https://parsnip.tidymodels.org/reference/boost_tree.html#:~:text=boost_tree()%20is%20a%20way,when%20creating%20the%20tree%20models> ( boost\_tree guide)
2. <https://rdrr.io/cran/parsnip/man/boost_tree.html> ( boost\_tree parameters explained)
3. <https://www.kaggle.com/matleonard/feature-generation> (guide to new features )
4. Ways of automatically tuning parameters
   1. <https://juliasilge.com/blog/xgboost-tune-volleyball/>
   2. <https://www.r-bloggers.com/tidymodels-and-xgbooost-a-few-learnings/>
5. <https://en.wikipedia.org/wiki/Hosmer%E2%80%93Lemeshow_test#Pearson_chi-squared_goodness_of_fit_test> (Hosmer-Lemeshow Test)
6. Read the pdfs with the names MADCAP\_explained, MADCAP\_example