Data Challenge

By: Brody Erlandson, Entong Li, Jiwoong Kang

Preprocessing The Data

Given there is a clear temporal nature to this data, we wanted to capture some of this complexity in our dataset. Given that every patient has a different amount of test's at different times, we wanted to create a few singular statistics that help capture three things: how they are overall in this test, what are the extremes of this test, and how do they change over time with this test. The first and second are easy to deal with, for the mean captures how they did overall and the max and min captures the extremes. One thing that would help capture the change over time is range, so we took this too. However it only captures the largest change, for if the change in the test is non-linear and ends back where it started; then, the range is only capturing the change in extremes. So we also added a statistic called change start to finish (changeStoF). This is the last test subtracted by the first test. Along with range it can help capture how the test changed over time. The last statistic we added was the number of tests. The idea is that if someone has been tested a lot, then something is probably wrong. There are a lot more statistics that probably could have been engineered, but we felt that it was a good coverage. After running the program to create the csv with these variables of each test, we had about 230 features.

EDA and Data Imputation

Given we know there is a lot of missing data, we wanted to look into what has the most missing values and the percentage missing. As we found that the percentage of NA values will affect the outcomes, we think dropping the tests that have a high percentage of NA values is not a good decision, and we need to perform the data imputation to replace the NA values. We consider four ways to impute the data: 1) replaced by mean/median computed based on outcome = 0; 2) replaced by mean/median calculated based on AdmissionType; 3) replaced mean/median calculated based on gender, age range, and outcome = 0; and 4) replaced by mean/median computed based on both AdmissionType and outcome = 0. After comparing the performance of models from each of these four imputed datasets, we found that there is no significant difference, thus we choose to use the third way to impute the test data then apply the final model.

Models and Model Selection

We consider three types of models totally, they are: Neural Networks, Random Forest/Boosting, and Support Vector Machines. We divided the train data into a training set(70%) and a test set(30%) randomly. For each model, we first apply the model on the training set, then tune the parameter by finding which sets of parameters could give us the lowest test error and highest AUC values. Overall, we choose the final model that has the highest AUC values and test accuracy among three models. The details of modeling is given below. Note that the Neural Network is the model that performed best with AUC and BER, so this is the model we decided to go with.

- **Neural Networks:**

We started with a clean 5 inner layer model, with nodes at 128, 64, 32, 16, and 8. This model unsurprisingly overfit, so we started with simplifying the model a bit. We then found with a 3 inner layer model at 32, 16, and 8 nodes were doing a little better; however it was still overfitting so we decided to add drop out to our model. While this was still overfitting, it was much better than before. It didn't seem to capture the model past about .3 accuracy. So we decided to try regularization on the models also. We found that L1 regularization did not perform well, this is probably because it actually zeros out features. So along with drop out, we include L2. After some playing around, we found we got some of the better results with the following model: 2 inner layers with 32 and 16 nodes with drop out at .4 and L2 regularization; a one node output layer; and a learning rate of .0001 on the adam optimizer and the binary crossentropy loss function. We had to decide how many epochs to run or to do early stopping with a tuned patience parameter; after a 5 fold cross validation, we found the early stopping with a patience of 15 and a validation set of .2 to be the best performing. With an average BER of about .32 and AUC of about .68.

- **Random Forest/Boosting:**

As we also want to perform data reduction, we first apply the random forest to training set without setting the parameters, and use the subset of training set, which have only the first 100 variables that have both high MeanDecreaseAccuracy and MeanDecreaseGini for random forest, or have high relative influence for adaboosting, to fit the same random forest again to see whether it could give a higher test accuracy. As the subset does not give any significant improvement, we decide to use the whole training set to tune the parameter. Lastly, we found that the best random forest model will be the one has ntree=100, nodesize=30, which has test error 0.26798 and auc value 0.66231, while the best adaboosting model will be the one has interaction.depth=9, shrinkage=0.1, which has test error 0.2599 and auc value 0.6811.

We also consider to fit the model based on different AdmissionType. We found that after filtering by AdmissionType, both RF and adaboosting will give the higher value of AUC, around 0.7234, and lower value of test error, around 0.2689, for AdmissionType=1 group, while both models will give lower AUC, around 0.5474, but lower test error, around 0.2059. Overall, fitting models based on AdmissionType does not improve our prediction accuracy.

- **Support Vector Machines:**

We employ polynomial and radial kernels to build classifiers. To search hyperparameter over a grid space, four different costs and gamma values are investigated. For a polynomial kernel, we tune upto third order polynomial. From the tuning, the radial kernel with cost=1 and gamma=0.01 results in a best test BER of 0.3545179 and AUC value of 0.6610953.