Daniel Green
FST374
6 August 2020

Thyroid Buddy Brief Report

Please see the `eda-and-cleaning.ipynb` notebook for detailed comments of decisions.

See the code of `model.ipynb` for details on that side.

Data Processing

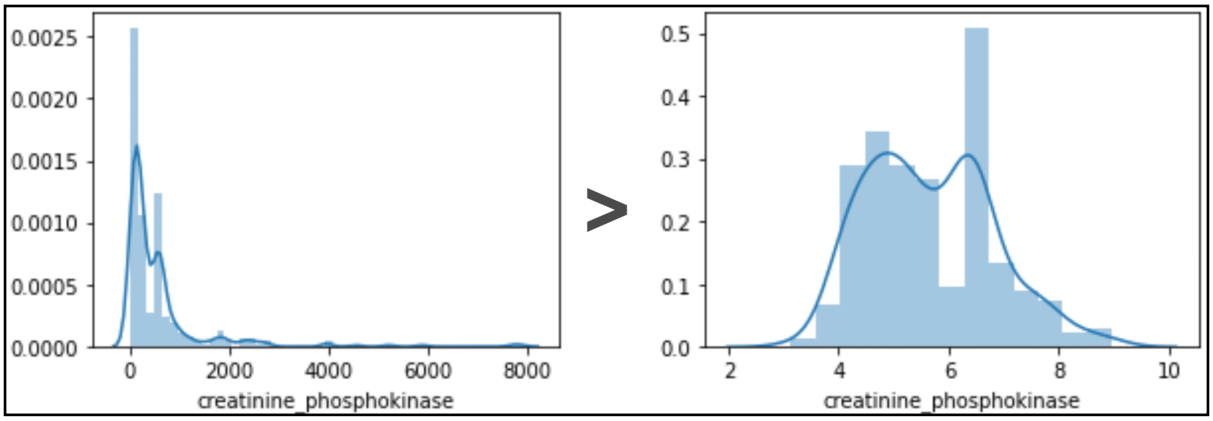
I started by checking for duplicate values. None were present, but if they were, I would have investigated why and remedied them accordingly by checking for errors or removing them.

I next checked for null values, again with none present. If they were found, I would have decided on a method of imputation, and as a last resort dropped them.

I next checked the data types and identified `age` and `platelets` that should have been integers but were floats due to some rounding or data entry issues. They were infrequent enough, constant enough, and close enough to a whole number to round and convert.

Next, data ranges were checked compared to those reported in the paper (since that data was available). All values were within range except two. The first appears to be a typo on their end, and the second was a simple division to alter the scale of `platelets`.

I then checked the skewness of all columns, with a threshold of 0.5. This value was chosen as a liberal value to include even moderately skewed columns. The skewed columns were rescaled using a natural logarithm , bringing them closer to a normal distribution. I originally used boxcox, but this has some issues resulting in large lambdas, so I went back to log.



Before and after skew fixing KDE plots.

After dealing with scaling, I identified outliers for all columns. I decided on using the 1.5*IQR method, as it gave solid results. Being pioneered by Tukey, it's standarized and also robust (resistant to error based on deviations like skewness). I could have used others here like z-score, but IQR worked just fine and I had no reason to do so with this small dataset. For each column, I calculated how many outliers were present, how many of those outliers were deaths, and how many of those deaths made up the global death count. I also then, for each outlier, calculated their distance from the mean of the parent column without them contained to get an idea of their distance (this was a little experimental but useful). I also plotted a traditional box plot of the column for visual inspection. While I'm certain with scrutiny I could have imputed/replaced/removed some outliers, I decided that the outliers I were seeing were acceptable. Given the small dataset, the 'outliers' may very well just be low probability values, and in fact may contribute to death.

I then checked the balance of classes between deaths and non-deaths. I'll cover that in the next section later on, so please look below for more information.

I then did some feature selection with a few methods. Firstly I tried `SelectKBest` using ANOVA as the target is binary and the predictors continuous. I then tried Recursive Feature Elimination which resulted in the same outcome. I finally tried Pearson's/Spearman's correlations, ignoring the sign of correlation and focused on the intensity, which kind of works given that we can represent our binary problem as two numbers. Regardless, '`age`', '`ejection_fraction`', and '`serum_creatinine`' were the top features in all methods. I also checked between these features for collinearity, but nothing stood out.

I followed this up with little exploration into the chosen variables in respect to the target variable values (died and not died) to see if there were any trends.

Data Imbalances

The data was quite imbalanced with only 67 deaths and 142 survivals. It's not as imbalanced as fraud, which requires other techniques, but it's not split as evenly as I'd like.

To solve this I implemented two approaches: oversampling and synthetic generation.

Oversampling was implementing using sklearn's `resample` function to oversample those that died to match those that didn't. This is essentially sampling with replacement.

I tried another approach using imblearn's `SMOTE` algorithm, which instead of repeatedly sampling the existing data synthetically creates new data based on clusters and neighboring values. This method also balances the values equally (in my case).

I found that even though SMOTE introduce 'fake' data, it had the best performance impact.

Modeling

For training, I split the data 30%. I usually split 20% but I had better results with the former.

For metrics, I decided on accuracy, F1 score, and ROC AUC, with F1 being used as the main metric as it is perfect for binary classification, which is what we had.

I tried a variety of classification models with varying results. However, the technique used for each was the same. I firstly create a pipeline with a scaler to normalize the data, followed by the model. I then use a randomized cross-validation search space (to save on performance sometimes) with hyperparameter tuning in place (which were discovered mostly through previous experimentation).

The trained models were saved as pkl files by exporting their search space objects (which in turn contain the pipelines, which in turn contain the models). This can be loaded later.

Deployment & Frontend

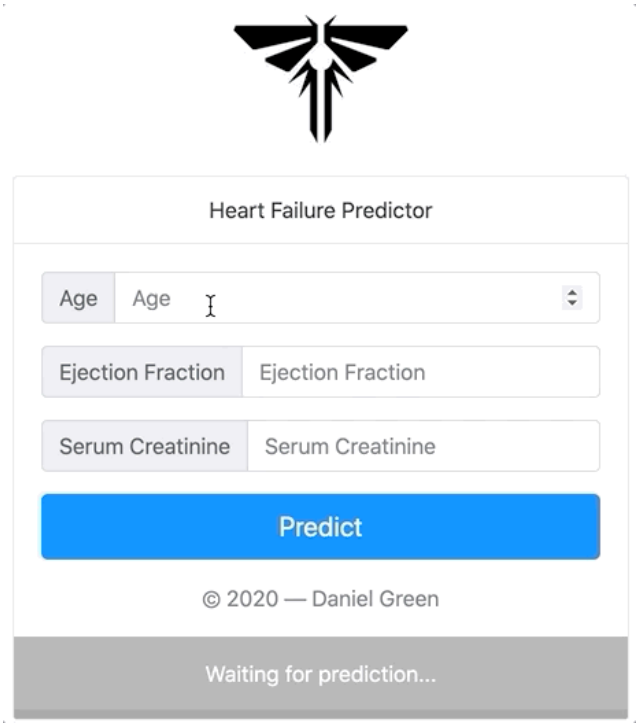
I created a complete deployment of the model (built to be swappable by simply replacing a file) as a RESTful API as well as a frontend to go with it in Flask.

The API uses a POST request to the `/predict` route expecting JSON and returning JSON. Please see the README.md for more information on how this all works.

I also built a frontend that uses the API (rather than communicating directly, which I could have done) to demonstrate that any application could indeed use this API. I decided not to add authentication to this simple demo, but it isn't too hard to add on top if necessary.

Disclaimer

The absolute vast majority of this code, including the EDA, modeling, API, and frontend, are my own code. Where I have used any code from outside sources, I have explicitly provided a link to it. I am happy to walk through all of the code in detail if necessary.



Frontend screengrab (please see the README for an animated version).