Final Project Write-up

Caleb Austin and Alex Moree
CSCE 421-500 - Spring 2023
Made with a Template by: Mortazavi

Abstract

It is often useful for doctors to be able to predict someone's odds of death when they enter the emergency room, so that they can prioritize care. In this project we are seeking to create an automated system that can take in someone's health information and use that to predict someone's odds of mortality while at the hospital. The consists of some basic information about the patient, such as age and weight, as well information gather about the patient through labs or by the nurse. Each of these different labs or nurse measurements are on different lines connected by a patient id number. Our model combines all the data about each patient into a single data point and builds gradient boosted trees to predict the patient's odds of mortality. Our results with this model have been good; we got more than 92% accuracy on our testing data.

1 Introduction

In this project we were seeking to create a model that would provide medical professionals with a way of accurately determining the odds that any given emergency room patient would survive their hospital stay. This allows them to give more attention to those who are mostly likely to die, so as to minimize deaths. To build our model we chose to use gradient boosted trees. We originally used an RNN, but only got mediocre results; so we tried to use gradient boosted trees, because they tend to provide good results on large datasets with a lot of features. We have since gotten good results. Specifically, we got more than 92% accuracy on our test dataset.

2 Your Method

For our model we first compressed each patient's health records into a single data point. Then, we build gradient boosted trees using those data points. Those trees can now be used to predict the mortality of anyone who's health records have been put in the same format.

2.1 Data Preprocessing

For preprocessing our model first converts the `patientunitstayid` to an integer to avoid issues when working with it. Then, we drop any data points with the `nursingchartvalue` of `Unable to score due to medication`; those data points won't work within the model and don't have much information anyway. Next, we convert all the `nursingchartvalue` to floats (or NaN if they cannot be converted) so that they can be processed as numbers (without lossing all the data contained in data points without this feature). We create a copy of the original dataset with the `admissionheight`, `admissionweight`, `age`, `gender`, `patientunitstayid`, and `unitvisitnumber`. This copy is what we will ultimately use with the model. Creating a copy is not necessary, but simplifies the process of preprocessing the data by allowing us to build the data from the ground up rather than having to modify an already existing dataset. After creating the copy, we one-hot encode the gender feature and replace `> 89` with 100 in the age column. This prevents issues with cases where the gender feature isn't filled in and makes sure the age column is always a number so that `> 89` cannot create an error. Next, we drop duplicates in the `patientunitstayid` column and make the `patientunitstayid` column the index. This makes it so that each patient is just one data point to simplify the model and make it so that data on a patient is indexed by their id.

After that, we proceed to create data columns, copy other pieces of the original dataset and modify them in minor ways, before pulling the data out of them and putting it into our first copy. For the first round of this, we create a data column in first copy called `cellattributevalue` and set it to `None`. This provides us a place to fill in those values, and will have a default value if there is nothing. We create a second copy of the dataset with the `cellattributevalue`, `celllabel`, `offset`, and `patientunitstayid`. We drop any data points in that copy with missing values to keep the data clean. Then, we encoded `< 2 seconds`, `> 2 seconds`, `normal`, `hands`, and `feet` as numbers to make it easier for the model to work with them. After that, we copied that value into the `cellattributevalue` feature in the data point in the first copy with the same patient id. For the second round, we create `pH` and `glucose` columns and fill them with `None`. Then, we create a third copy of the dataset with the `labname`, `labresult`, `offset`, and `patientunitstayid` features. Next, we drop any data point with missing values in the third copy. After that, we copy the `labresult` into the `labname` feature in the data point with the same patient id in the first copy. If there is already some number in the `labname` feature, the smaller value is used. For the final round, we add the columns `GCS Total`, `Heart Rate`, `Non-Invasive BP Diastolic`, `Non-Invasive BP Mean`, `Non-Invasive BP Systolic`, `Invasive BP Diastolic`, `Invasive BP Mean`, `Invasive BP Systolic`, `Respiratory Rate`, and `O2 Saturation` and fill them with `None`. Then, we create a fourth copy of the dataset with the `nursingchartcelltypevalname`, `nursingchartvalue`, `offset`, and `patientunitstayid`. Next, we drop any data points that contain any missing values. After that, we copy the `nursingchartvalue` in the `nursingchartcelltypevalname` feature in the data point in the first copy with the same patient id, unless the is already a smaller value there.

Finally, we convert all the data into floats and fill in any missing values with the average value for that feature. This prevents any errors from missing values or values being an unexpected type.

2.2 Model Design

Our model consists of a set of gradient boosted trees. We choose to use gradient boosted trees, because they tend to work well on large datasets with lots of features. In particular we used sklearn's GradientBoostingRegressor with 600 boosting stages. The trees generated during each of these stages were limited to a depth of 2, each leaf had to have at least 4 samples, and any internal node could not be split unless at least 2 samples reached it. We used at learning rate of 0.01 and a random seed of 42. These hyperparameters were originally derived from hyperparameter tuning; except the random seed, because that is random. We subsequently performed some manual testing to find some slightly better hyperparameters.

2.3 Model Training

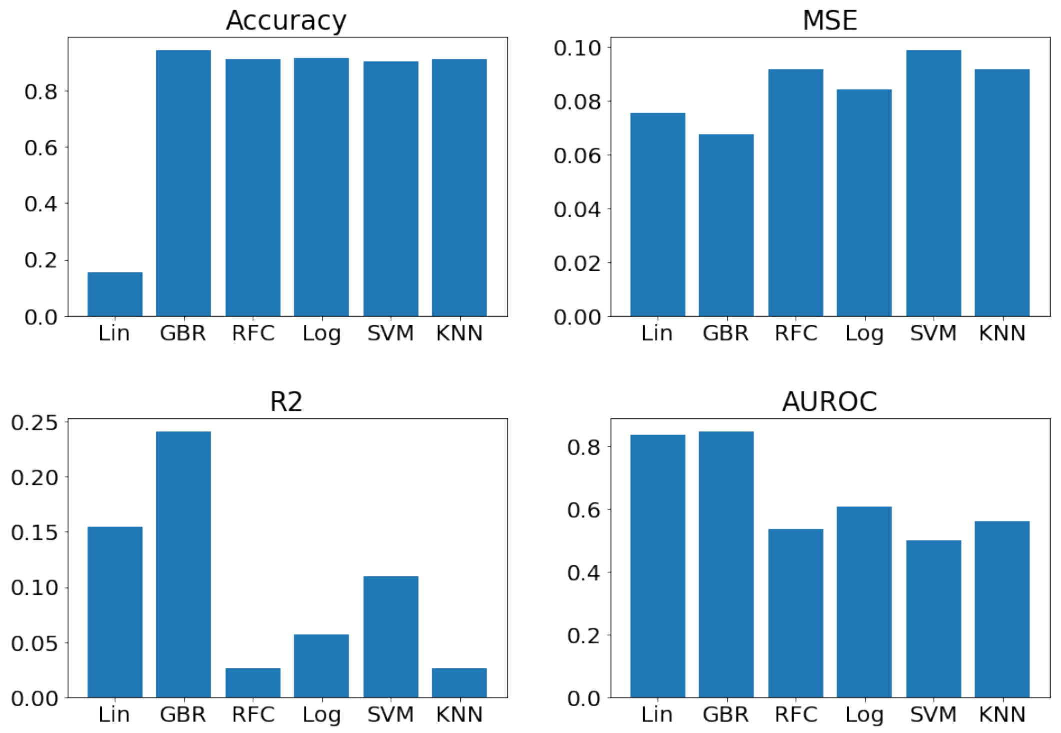
We simply used sklearn's fit method to train our model. Internally, that works by first selecting the average of the expect value. Then, for each of the 600 boosting stages, it adds the set of trees with a new tree conformant to the parameters given above. Each new tree selected to minimize the loss function, mean-squared error in this case. However, each new tree is only optimized on a random half of the data. Only training each tree on half of the data prevents overfitting.

2.4 Hyperparameter Tuning

We conducted hyperparameter tuning using sklearn's GridSearchCV. GridSearchCV takes in the GradientBoostingRegressor model and a dictionary defining the hyperparameters we want it to test. We had it conduct 5-fold cross validation on every possible combination of the hyperparameter we gave to it. We had it try 300, 400, 500, 600, 700, or 800 boosting stages; a learning rate of 0.1, 0.01, or 0.001; a maximum tree depth of 1, 2, 3, 4, or 5; minimum numbers of samples to split a node of 1, 2, or 3; minimum number of samples to split a leaf node of 3, 4, 5, or 6; and subsample sizes of 0.25, 0.5, 0.75, or 1 for building each tree. We subsequently did some testing to get better results. This may have overfit our results to the test data a little, but our accuracy would still be above 90% without that. The best hyperparaters found with the cross-fold validation were a learning rate of 0.01, max depth of 4, min leaf samples of 5, min samples to split of 2, and 400 boosting stages.

3 Results

The results of our model have been quite good. To demonstrate we preprocessed our data as before and split off 20% of the data to be validation data. We trained numerous different models on the training data and tested it on the validation data using four different metrics.



We abbreviated the names of the different models to make them easier to read. You can use the table below if you are unsure as to what any model is.

Abbreviation	Model
Lin	Linear Regression
GBR	Gradient Boosting Regressor
RFC	Random Forest Classifier
Log	Logistic Regression
SVM	Support Vector Machine
KNN	k-Nearest Neighbors

As you can see our model preformed best on every metric except mean squared error. It achieved more than 92% accuracy on the hidden test data.

4 Conclusion

The hardest part of this project was explicitly the processing of the dataset and subsets. This took a lot of online searching and out-of-the-box thinking that one would normally oversee to create the end result. We think some of the difficulty was due to our unfamiliarity with medical data. If we where more accustomed to this type of data this process would have been simpler. Splitting the data into subsamples was by far the most effective strategy for improving accuracy. Then, we just needed to find a model competent with large numbers of features which was why we chose Gradient Boosting. It was actually by accident, as the first option was XGBoost, but we encountered import errors, so we tried the Random Forest Classifier, but it overfit the data way too much and resulted in low accuracy scores. Lastly Gradient Boost was chosen, because it was referenced in some of the lecture material and we decided to give it a shot. This endeavor proved to be quite fruitful. Gradient Boost is good for large feature sets because it can handle high-dimensional data and feature interactions, leading to better predictive accuracy resulting in what you see now.

5 References

1. ChatGPT

   ChatGPT was used to help us learn how to work with convoluted datasets like what we had here.

2. Stack Overflow

   We referenced conversations on Stack Overflow to help us understand how gradient boosting works and to ensure that we were using correct syntax.

3. sklearn

   We referenced the documentation to both get a better understand of how the model we were using worked and to see what options were available for given specific models and functions.