Final Project Write-Up

Arya Rahmanian, Johnny Le, Ismat Syah Imran
Department of Computer Science
Texas A&M University
College Station

1 CSCE 421 - 500 - Spring 2023 - Mortazavi

Abstract

This report aims to predict mortality for patients admitted to intensive care units (ICUs) within hospitals. The data utilized in conducting the study is a subset of the eICU database, which includes over 2,000 unit stays selected from 20 of the largest hospitals in the United States between 2014-2015. By utilizing machine learning, we hope to advance technology in predicting the mortality of these patients. We utilized feature engineering techniques to preprocess the data, followed by a supervised learning approach with a Gradient Boosting classifier. We achieved the best performance utilizing this model with a score of 0.90019. Based on the results, our work suggests that machine learning and technology can play an effective role in predicting ICU mortality and aid in clinical decision-making for ICUs.

2 Introduction

The goal of this study was to build a supervised model that is capable of accurately predicting the mortality of patients admitted to the ICUs in hospitals. ICUs consume a ton of resources from hospitals just to hold a single patient, so assisting healthcare providers in informed decisions about patient care and resource allocation is crucial in maximizing the survival rate of all patients. Additionally, it can measure the performance of hospitality in the hospital to identify areas for improvement in patient care.

3 Your Method

This section describes the data pre-processing, model design, model training, and hyper-parameter tuning process for the model. The project involved creating a model to predict the probability of death of a patient in hospitals' Intensive Care Units. The data was pre-processed by aggregating patient data, cleaning and encoding categorical data, and dropping unnecessary columns. The team attempted to use a recurrent neural network (RNN) but switched to a tree-based approach due to difficulty in converting the data to a 3D shape. Achieved the best results using a gradient boosting classifier with the "log_loss" loss function and a learning rate of 0.1 and much more.

3.1 Data Preprocessing

From the initial looks at the data, we noticed that if we were going to train a model, a lot of encoding and aggregation had to be done before the training. Our initial goals in creating a model were through a Recurrent Neural Network, specifically an LSTM model, given the nature of the time series data where each patient can have multiple records. However, due to the way we pre-processed the data and struggled to convert the data into an acceptable 3D shape, we switched our pre-processing plan.

Our plan was to aggregate or combine all the patient data into one record. This mean that if a patient had multiple observation of heart rate measurements then we would take the average. The reasoning behind our choice is that we fail to get the dataset training beyond matching the length of the train_y data which had a shape of (2016,1). Before combining the data, we also noticed that some columns, which seemed numerical by default, held string values: "> 90" and "Unable to score due to medication." So we quickly cleanse those values to 90 and 0 respectively so that we can assign the datatype of those columns to float for future processing. Additionally, we dropped a couple of columns: ['Unnamed: 0', 'cellattributevalue', 'celllabel', 'labmeasurenamesystem', 'labname', 'labresult',], due to the sparse amount of data there was on them or our judgment that they would not be significant. Next, we ran our data through an encoding function we made that essentially takes all the categorical data and creates unique columns for the data (hot encoding) to be 0 or 1.

As you tell, the data has a bunch of new columns due to the hot-encoding, which is good because it will be easier to distinguish those features separately compared to when they were mixed into one column. However, there are a lot of zeros for the nursing chart as not every record has a value for each type of chart. That said, we decide to aggregate the data using the mean of the nursing value that did not have the value of 0.

From there our final number of records matched our y_train's number of records and we were ready to build our model and train.

3.2 Model Design

Our original plan was to create an LSTM model to train on our data, but after some initial attempts, our team wasn't able to get good enough results with our models. We believe this is due to not preprocessing the data in the right way to feed into the NN for the model to train on. This led to our team trying a tree-based approach. First, we saw decent results with Decision Trees, around 60%. From there, we improved upon that by using Gradient Boosting. With

nursingch	offset	patientun	unitvisitn	ethnicity_	ethnicity_	ethnicity_	ethnicity_	ethnicity_	ethnicity_	gender_F	gender_M	nursingch
	0	141764	2	0	0	1	0	0	0	1	0	0
182	256	141764		0	0	0	0	0	0	0	0	0
80	269	141764		0	0	0	0	0	0	0	0	0
181	196	141764		0	0	0	0	0	0	0	0	0
98	331	141764		0	0	0	0	0	0	0	0	0
132	269	141764		0	0	0	0	0	0	0	0	0
102	196	141764		0	0	0	0	0	0	0	0	0
103	256	141764		0	0	0	0	0	0	0	0	0
166	269	141764		0	0	0	0	0	0	0	0	0
143	196	141764		0	0	0	0	0	0	0	0	0
128	166	141764		0	0	0	0	0	0	0	0	0
106	166	141764		0	0	0	0	0	0	0	0	0
173	166	141764		0	0	0	0	0	0	0	0	0
107	76	141764		0	0	0	0	0	0	0	0	0
87	76	141764		0	0	0	0	0	0	0	0	0
153	331	141764		0	0	0	0	0	0	0	0	0
78	331	141764		0	0	0	0	0	0	0	0	0
128	76	141764		0	0	0	0	0	0	0	0	0
98	196	141764		0	0	0	0	0	0	0	0	0
103	331	141764		0	0	0	0	0	0	0	0	0
102	269	141764		0	0	0	0	0	0	0	0	0
176	76	141764		0	0	0	0	0	0	0	0	0
133	256	141764		0	0	0	0	0	0	0	0	0

Figure 1: Data After Hot Encoding

patientun	offset	admission	admission	age	unitvisitn	ethnicity_	ethnicity_	ethnicity_	ethnicity_	ethnicity_	ethnicity_	gender_F	gender_M	nursingch	nursingch	nursingch	nursingch	nursingch	nursingch	nursingch	nursingch	nursingch	nursingch
141764	5176	157.5	0	87	2	0	0	1	0	0	0	1	0	0	107	0	0	0	92.66667	124.3333	171.8333	0	0
141765	124902	157.5	46.5	87	1	0	0	1	0	0	0	1	0	0	93.70588	0	0	0	74.89583	101.7021	152.375	96.71429	21.28571
143870	39851	167	77.5	76	1	0	0	1	0	0	0	0	1	0	46.22222	39.34615	0	111.3462	53.85714	0	105.9286	97.04	27.66667
144815	57163	172.7	60.3	34	1	0	0	1	0	0	0	1	0	0	88.375	0	0	0	75.04545	90.2381	116.9091	98.80769	19.625
145427	318257	177.8	91.7	61	1	0	0	1	0	0	0	0	1	0	69.77273	0	0	0	64.41538	0	117.7692	95.69231	16.15385
147306	18650	157.5	0	55	2	0	0	1	0	0	0	1	0	0	75.6875	66.7	0	139.9	69.5	95.25	134.0625	95.5	16
147307	25851	157.5	72.5	55	1	0	0	1	0	0	0	1	0	0	88.60465	65.09756	0	136.8537	70.03704	94.57692	136.0741	93.75	16.60714
147784	400301	154.9	95.6	60	1	0	0	0	1	0	0	1	0	0	89.58442	0	0	0	71.52778	102.6515	153.3056	94.74074	24.73585
148611	74284	182.9	91.8	28	1	0	0	1	0	0	0	0	1	0	89.67742	0	0	0	75.25	98.40909	139.2143	96.03846	24.2963
149432	3649	165.1	0	34	2	0	0	1	0	0	0	1	0	0	95.57143	0	0	0	66.42857	79.28571	102.8571	98.28571	18
149433	-1853	165.1	60.7	34	1	0	0	1	0	0	0	1	0	0	100.5	0	0	0	80.33333	94	123.3333	98.33333	27.25
149713	49548	157.5	58.5	90	1	0	0	1	0	0	0	1	0	0	69.125	0	0	0	64.66667	84.17391	111.9167	94.18182	25.95455
149714	42334	157.5	0	90	2	0	0	1	0	0	0	1	0	0	70	0	0	0	68.375	92.625	128.875	95.4	24.14286
151867	406163	172.7	0	44	1	0	0	1	0	0	0	0	1	0	89.21429	0	0	0	75.97778	0	127.6667	96.12121	20.34483
153972	123441	172.7	0	63	1	0	0	1	0	0	0	0	1	0	76.45455	0	0	0	65.60714	0	126.6607	95.84848	17.61538
155961	470369	157.5	120.1	57	1	0	0	1	0	0	0	1	0	0	82.42202	0	0	0	58.1129	0	114.3871	95.11607	17.54955
156308	296369	172.7	86.18	87	1	0	0	1	0	0	0	0	1	0	88.26667	0	81	0	59.8	0	122.8333	95	20.96429
156906	253989	160	115.2	52	1	0	0	1	0	0	0	1	0	0	83.08333	0	0	0	82.0339	115.2586	169.2881	94.20779	21.70909
157016	-1439473	162.6	63.5	23	1	0	0	1	0	0	0	1	0	0	88.42975	0	97	0	79.53012	0	124.253	98.47805	18.08696
157427	600776	180.3	86.2	73	1	0	0	1	0	0	0	0	1	0	63.59763	0	110	0	59	0	121.593	96.85047	17.88889
158057	285976	193	0	39	2	0	0	1	0	0	0	0	1	0	73.94444	0	0	0	73.52632	0	128.8947	97.31579	18.52632

Figure 2: Data After Preprocessing

Gradient Boosting, we saw our best results, around 90% accuracy, and decided to continue from there, using the "log_loss" loss function.

3.3 Model Training

After processing the data, we split the preprocessed x and y training data-frames into training and validation sets. We then were able to instantiate the Gradient Boosting Classifier model and train it with our data and test its accuracy.

3.4 Hyperparameter Tuning

With the model we selected, our team tested various types of loss functions provided but found that the exponential loss function provided the best results with our data.

Same with the learning rate, after messing around with the different values from 1 to 0.0001. They all ended up being similar but, once again the default learning rate of 0.1 proved to be the best.

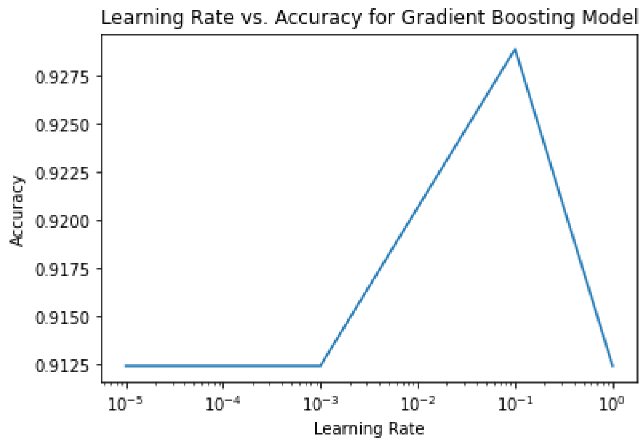

Figure 3: The image above the different accuracy scores for various learning rates tested

Our team also tested a range of values for min_samples_split, min_samples_leaf, and max_depth. None proved to be more successful when tuning so they were kept as default.

Additionally, we attempt to utilize GridSearchCV in searching for the best hyper-parameters; however, this process is partially unsuccessful as it seems to be hanging on outlying performance numbers. Our hypothesis is that the parameters we provided are not optimal in searching for the best hyper-parameters

Additionally, when splitting the training set into validation, there wasn't much vary in the accuracy results between 0.2 to 0.5 validation data size.

4 Results

The results of our model are outputted into a .csv file as shown in Figure 4 and achieved a 90% accuracy. Changing the hyperparameters created models that had accuracy rates as low as 72% so it is entirely possible that models that scored lower such as the baseline submission used the same method and merely had different hyperparameters. Our initial models produced using recurrent neural networks were able to reach at most scores of 66% in part because we were unable to properly convert the dataset into a 3D shape required. Our initial gradient boosting model however achieved 86% accuracy and we

patientunitstayid	hospitaldischargestatus
151179	0.212392707
151900	0.020897101
152954	0.023156563
158056	0.02285204
159411	0.014534598
160312	0.012222277
162502	0.016573331
165173	0.017859513
166853	0.016730681
167185	0.013721984
168726	0.051099508
169588	0.611300518
175110	0.016808584
175243	0.014455369
176050	0.012232903
176730	0.047683384
177689	0.024031361
179028	0.012687375
181906	0.028840049
185387	0.011809263
187823	0.019491943
192233	0.012018842
193665	0.014488859
197616	0.022337378
197921	0.12554765
203970	0.08531289

Figure 4: Output from evaluation dataset

5 Conclusion

Our model choice was to focus on gradient boosting as we were able to achieve the best results even compared to more complex machine learning techniques such as recurrent neural networks. Gradient boosting relies heavily on a robust loss function so we decided not to potentially overcomplicate or overfit to our dataset by using log_loss which is an implementation of cross-entropy loss. For our preprocessing we chose to aggregate the patient data into only unique patients thereby reducing the amount of computation required. We also chose to remove features that had sparse data including all of the lab results. We believe that removing these features allowed our model to focus on more predictive features and not cater to features that relatively few entries had a value for. To improve our method in the future we may focus on using a different loss function as this would have the greatest impact on a gradient boosting algorithm.