eICU mortality prediction

Minh Dao Nguyen

Abstract

The best model that I could develop to predict patient in-hospital mortality given a set of medical data has an accuracy score of 88%. This paper describes the process of how I develop the model using various Machine Learning techniques. The final model that I built use features dropping techniques, AdaBoost model, and scalling method.

1 Introduction

This paper describes the process which I have taken to build a model to predict the in-hospital mortality patients using eICU data. The data contains medical data of a patient when they visit the ICU such as vital sign measurement, age, gender, etc. I built a Machine Learning model that can predict, as best as it can, the in-hospital mortality of a patient based on all the medical data from that patient. I used multiple Machine Learning techniques such as features selection, hyperparameter tuning, as well as multiple different Machine Learning models to best predict the in-hospital mortality of a patient. This paper discuss and showcase all the techniques and process that I have used in developing the best performing model that I can build.

2 Method

The methods that I used will be break down into portions: Data Prepossessing, Model Design, Model Training, and Hyperparameter Tunning. In each steps, the paper discuss more details on techniques, findings, and reasoning behind all the steps I made.

2.1 Data Preprocessing

2.1.1 Data

The dataset is a subset of the eICU dataset. It contains deidentified, clinical health data for over 200,000 admissions to intensive care units across the United States, between 2014-2015. The database includes vital sign measurements corresponding to each visit occurrence. Data is collected through the Philips eICU program, a critical care telehealth program that delivers information to caregivers at the bedside. This open access demo allows researchers to ascertain whether the eICU Collaborative Research Database is suitable for their work. It includes over 2,500 unit stays selected from 20 of the larger hospitals in the eICU Collaborative Research Database.

2.1.2 Removing empty measurements

While processing the data, I found out that there are two measurements that are missing values, glucose and GSC total measurement. Therefore, I created a loop that loop through two type of measurments ’labname’ and ’nursingchartcelltypevalname’. If any of the measurement values are empty, the row is removed. Alternate methods such as K neightboring could be implementing to fill in the NA value. But I found out that so far, there are only 7 lines total that are missing values. Therefore, removing them would not have too much of a significant impact comparing the rest of thousand lines of data.

2.1.3 Grouping the Data

First I will be grouping the data by their 'patientunitstayid'. The index of each row of the dataframe now ill be the patientunitstayid.

2.1.4 Creating columns for each measurements

Then I will loop through the 'labname', 'nursingchartcelltypevalname', and 'celllabel'. All of these holds the name of different types of measurement done on the patients. Therefore, I will take out the unique values from each of these columns and makes those values a new columns.

The measurements in 'labname' are ['glucose', 'pH'].

The measurements in 'nursingchartcelltypevalname' are ['GCS Total', 'Heart Rate', 'Invasive BP Diastolic', 'Invasive BP Mean', 'Invasive BP Systolic', 'Non-Invasive BP Diastolic', 'Non-Invasive BP Mean', 'Non-Invasive BP Systolic', 'O2 Saturation', 'Respiratory Rate'].

Lastly, the measuremnts in 'celllable' is ['Capillary Refill'].

After the new columns are created. I input the data into the apporiate columns: 'labresult' values will go with the measurments in 'labname', 'nursingchartvalue' values will go with the measurments in 'nursingchartcelltypevalname', and 'cellattributevalue' values will go with the measurments in 'celllabel'. I simply added the data from the corresponding index of the list of the measurement value to the measurement types columns. I can only do this if the len of the list of the measuremnt value is the same length as the type of measurement. Therefore, the cleaning data and filling in NA value is crucial to make this step as simple as possible.

2.1.5 Clean data

Fill in NA value: I filled in default data for some missing data in each features. The default data here means that average means for the population of the data. I only apply this for features that I think wouldn't be important for the model. The following are what data I filled in for different features.

Features	data
gender	'Male'
ethnicity	'Caucasian'
age	'30'

Reduce list: The first step of further cleaning more of the data is reduce the list type of the data for features that have one measurements. This include 'gender', 'age', 'ethnicity', 'unitvisitnumber'. I applied a function called 'reduce_list', where it will take the first element of the list. Therefore, the function is only appropriate for features that only have one value.

Average values: For the measurements features, I calculated the average values of the measurement and input it to the features. I applied to the function 'avg _val' to the following features/columns: 'glucose', 'pH', 'GCS Total', 'Heart Rate', 'Invasive BP Diastolic', 'Invasive BP Mean', 'Invasive BP Systolic', 'Non-Invasive BP Diastolic', 'Non-Invasive BP Mean', 'Non-Invasive BP Systolic', 'O2 Saturation', 'Respiratory Rate','Capillary Refill'. The function 'avg _val' also automatically checked if the list is empty, if it does, then the function will automatically fill in 0.

Age features: In the age data, there are some data with '> 89' values for people that are older than 89. I applied the function 'change _age' to that 'age' columns, and the function changes every data with '> 89' to 90.

2.1.6 One-hot encode:

I used one-hot encode for non-numerical data. There are 3 features that are categorical data: 'gender', 'ethnicity', and 'Capillary Refill'. I used the 'pd.get _dummies' to creates a columns for each categorical data in each features, then I add those columns into the main data frame.
Before moving forward, I have to check if all the categorical data is presented, this was done after

examining the test data set. Because the training data set is bigger than the testing one, there will be more unique different categorical data in ’ethnicity’ and ’Capillary Refill’ features. Therefore, I manually added checking function for each of the categorical data in those features. If a column is missing, a new columns with that name will be added filled with 0. This step is needed in order for the testing data to have the same features as the training data.

2.1.7 Preparing for models:

In order for the data to be trained in models in the Sklearn library. Some of the requirements need to be met. First there has to be no NA value in the data set. Therefore, a quick fix to this is implementing ’fillna(0.0)’ function in the Panda library. This function fill any value with 0.0. Secondly, the columns/features name has to be a string. During the debugging process, I found out that some columns has _str as their columns name. Therefore, using the ’astype(str)’ funcstion and ’df.columns.’, I changed all columns name to String types. Lastly, every data entries need to be a float type. Therefore, for one-hot encode data, where they are automatically an integer, I changed them into float value using ’astype(’float’)’ function.

2.2 Model Design

What model did you choose? Why did you choose that model? What parameter did you use for the model if they weren’t tuned during your hyperparameter tuning?

I used 5 different type of model from Sklearn library: AdaBoost, Gradient Boosting, Neural network, Logistic Regression, and Supporting Vector Machine.

2.3 Model Training

I used ’GridSearchCV’ function in Skearn library to train model. This function automatically implement all hyperparameters that are passed to it with the given model. I passed in cross-validation value 5 into the ’GridSearchCV’ function. After the that, I will check the accurary of the each model using AUC score with the test data.

2.4 Hyperparameter Tuning

For tuning the model, I created different python library ’param _grid’, and I passed it into the "GridSearchCV’ function along with the model. I change the ’param _grid’ library for different models. I also drop different features that run the model again to see the change in results.

2.4.1 Hypertuning process for different Models

AdaBoost

```
ada = AdaBoostClassifier()
#Hypertunining
param_grid = {’n_estimators’: [50, 100, 200, 300, 400, 500, 1000],
              ’learning_rate’: [ 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1],
              ’algorithm’: [’SAMME’, ’SAMME.R’]}

#train with grid search
grid = GridSearchCV(ada, param_grid, cv=5)
grid.fit(X_train, y_train)
```

Gradient Boost

```
param_grid = {’n_estimators’: [100, 200, 300],
              ’learning_rate’: [0.01, 0.1, 1],
              ’max_depth’: [3, 5, 7],
              ’min_samples_split’: [2, 5, 10]}
gb = GradientBoostingClassifier()
grid_search = GridSearchCV(gb, param_grid=param_grid, cv=5, verbose=3, n_jobs=-1)
grid_search.fit(X_train, y_train)
```

Neural Network

```
# Create MLPClassifier
mlp = MLPClassifier()
# Define parameter grid for GridSearchCV
param_grid = {
    'hidden_layer_sizes': [(10,), (20,), (50,), (100,), (50, 50), (100, 100)],
    'activation': ['logistic', 'tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'learning_rate_init': [0.001, 0.01, 0.1],
    'max_iter': [100, 200, 500]
}
# Perform hyperparameter tuning using GridSearchCV
neural_model = GridSearchCV(estimator=mlp, param_grid=param_grid, cv=5)
neural_model.fit(X_train, y_train)
```

SVM

```
pipe = Pipeline([('scaler', StandardScaler()), ('svm', SVC())])
param_grid = {'svm__C': [0.1, 1, 10, 100, 1000],
              #'svm__kernel': ['linear', 'rbf', 'poly', 'sigmoid'],
              'svm__kernel': ['linear', 'rbf', 'poly'],
              #'svm__gamma': ['scale', 'auto', 1, 0.1, 0.01, 0.001],
              'svm__gamma': ['scale', 'auto', 1, 0.1, 0.01],
              'svm__degree': [2, 3, 4, 5, 6],
              'svm__coef0': [0, 0.5, 1, 2, 5, 10]}
svm_grid = GridSearchCV(pipe, param_grid, cv=5, n_jobs=-1, verbose=1)
svm_grid.fit(X_train, y_train)
```

2.4.2 Features Dropping

I dropped some features that I think is not important or unnecessary. This is done in order to avoid over fitting.

Ethnicity I dropped all the columns about ethnicity after the one-hot encoding. There are 6 features/columns about ethnicity, so I believe it would cause overfitting to the models.

```
df.drop(columns = ['ethnicity_African American', 'ethnicity_Asian',
'ethnicity_Caucasian', 'ethnicity_Hispanic', 'ethnicity_Native American',
'ethnicity_Other/Unknown'], inplace = True)
```

Gender I also dropped all the columns about gender after the one-hot encoding. There are 2 features/columns about ethnicity. I believe this is the second features should be de-selected in order for more important features to have more effects as well as avoid ocerfitting.

```
f.drop(columns = ['gender_Female','gender_Male'], inplace=True)
```

2.4.3 Features Scalling

Lastly, I used 'StandardScaler()' to scale the feature of the data set. This function standardize the features by scaling the features to its variance. In general, it helps with features that are important but doesn't have a high value. This is important for this data set especially when there are one-hot encoding. Some features like 'Capillary Refill' where each of its label has value 1 or 0. Other measurement feature like 'Hearrate' would have 80 - 100 value range. The 'Capillary Refill' will have less impact. Therefore, using feature scalling normalized the data set and make the all features has same impacts.
The only complication with 'StandardScaler()' is that it turns the dataframe into a np.array. Therefore, the test data has to be pre-processed along with the training data in order for them to have same features as well as features order.

3 Results

3.1 AUC score for different models with different tunnign method

Model	Default	Ethnicity drop	Gender drop	Scalling
AdaBoost	0.82	0.85	0.85	0.882
Gradient Boost	0.81	0.84	0.84	0.879
Neural Network	0.81	0.78	0.77	0.794
Logistic Regression	0.55	—	—	—
SVM	0.79 (low tunning)	—	—	—

From the results above, it seems like Boosting method is the best for this set of data. Features dropping as well as scaling data also helps with improving performance for boosting method. However, for neural network model, it seems like the top performance is 81%. The model decrease performance as the number of feature reduce as well as implementing scaling method. I stop with the first try of logistic regression because the base performance is so bad that putting more tuning effort seems not worth it. The resulted for SVM being undefined because my device could not handle running the SVM with a lot of hyperparameters tuning.

3.2 Final AUC score

Models	AUC score
AdaBoost	0.88221
Gradient Boost	0.87915
Neural Network	0.81857
Logistic Regression	0.55
SVM	undefined

4 Conclusion

In conclusion, AdaBoost is the best model with the AUC score of 0.88.
Since I have exhausted most of the hyper parameter tuning process. I believe that in order for the model to perform better, I either have to use a different Machine Learning model, or I have to make more change to the data pre-processing process. In data pre-processing, I did not use features 'offset'. I could use this features in order to increase my model performance. New features such as rate of change in vital measurements could be implemented using 'offset' data. However, it would require a lot more works to implement it is difficult to apply such calculation on such a large set of unsorted data, not to mention it could require overfitting by introducing new features. Therefore, introducing new features could results in old features being drop. In my conclusion, the model could perform betters, but it would require a lot of new features being made and possibly old features being dropped.