Heart Failure Prediction

Machine Learning

Carlos Navarro Naranjo

Executive Summary

More than 300,000 Americans will die this year of sudden cardiac arrest, when the heart abruptly stops working. These events happen suddenly and often without warning, making them nearly impossible to predict. Numerous studies have tried to implement machine learning as a useful tool that can help professionals with predicting heart failure mortality. People with cardiovascular disease or at high cardiovascular risk (due to the presence of one or more risk factors such as hypertension, diabetes, hyperlipidemia among others) need early detection and management wherein a machine learning model can be of great help.

This report uses clinical records of patients who had heart failures or complications collected during a follow-up period to see if patients survived or died during such time.

In this project I first analyze and preprocess the patients' data to later be used by machine learning models. The code developed in python takes care of cleaning the data to later apply two machine learning techniques:

- Artificial Neural Network Classification Model.
- Support Vector Machine Classification Model

These models are perfect for binary classification. Given a patient's clinical records, the models help predict mortality in patients that suffer from heart failure or other heart complications.

During the learning method phase of this project, models are optimized to gain the best possible results & performance. After improving the models' parameters and hyperparameters to obtain the best case for each model, the results and metrics obtained are presented to the reader in figures and tables. Such results are analyzed in the discussion section and used to make a final decision between which model makes more sense for the problem that this project tries to solve .

Dataset Analysis:

I am tasked is to create a model that accurately predicts the mortality caused by Heart Failure. For such task I used the dataset contained in a CSV file from Kaggle called: (heart_failure_clinical_records_dataset.csv) which holds the medical records of 299 patients who had heart failure, collected during their follow-up period, where each patient profile has 12 clinical features. The original dataset version was collected by Tanvir Ahmad, Assia Munir, Sajjad Haider Bhatti, Muhammad Aftab, and Muhammad Ali Raza (Government College University, Faisalabad, Pakistan) and made available by them on FigShare under the Attribution 4.0 International (CC BY 4.0: freedom to share and adapt the material) copyright in July 2017.

The first thing to do after examining the dataset is to create a function called cleanse_data that takes a Pandas' Data Frame as input and returns another Data Frame that was identical to the original except for the invalid rows that are removed using .dropna() to drop the rows that contain blank entries or non-numerical objects in any column. cleanse_data also eliminates rows that contained negative entries for any column by selecting the rows containing only values equal or bigger than 0 and takes that as our new data. This is because you can't have negative number of platelets in blood or negative age for example.

My dataset consists of:

- Features:
  - age: age of the patient [years]
  - anaemia: decrease of red blood cells or hemoglobin [Boolean]
  - high blood pressure: if the patient has hypertension [Boolean]
  - creatinine phosphokinase (CPK): level of the CPK enzyme in the blood [mcg/L]
  - diabetes: if the patient has diabetes [Boolean]
  - ejection fraction: percentage of blood leaving the heart at each contraction [percentage]
  - platelets: platelets in the blood [kilo platelets/mL]
  - sex: woman or man [binary]
  - serum creatinine: level of serum creatinine in the blood [mg/dL]
  - serum sodium: level of serum sodium in the blood [mEq/L]
  - smoking: if the patient smokes or not [Boolean]
  - time: Follow-up period (days)

- Response:

- death event: if the patient deceased during the follow-up period [Boolean]

[Boolean values: 0 = Negative (No); 1 = Positive (Yes)]

Using .count(), we know that after cleaning the data there are 299 data points which is enough to safely perform holdout validation. Using train_test_split I subdivide our dataset into 80% train data and 20% test data. Using .describe() we can look at our data's statistics:

count	count	mean	std	min	max
age	299	60.83	11.89	40.00	95.00
anaemia	299	0.43	0.50	0.00	1.00
creatinine_phosphokinase	299	581.84	970.29	23.00	7861.00
diabetes	299	0.42	0.49	0.00	1.00
ejection_fraction	299	38.08	11.83	14.00	80.00
high_blood_pressure	299	0.35	0.48	0.00	1.00
platelets	299	263358.03	97804.24	25100.00	850000.00
serum_creatinine	299	1.39	1.03	0.50	9.40
serum_sodium	299	136.63	4.41	113.00	148.00
sex	299	0.65	0.48	0.00	1.00
smoking	299	0.32	0.47	0.00	1.00
time	299	130.26	77.61	4.00	285.00
DEATH_EVENT	299	0.32	0.47	0.00	1.00

These is a useful insight to later understand that our features should be standardized. We can clearly notice that features like platelets or creatinine show significantly higher values all throughout the table in comparison with other features such as sex or smoking. These values serve as an important reference for our models. The above statistics tells us that features must be preprocessed.

Using the bar charts shown below we can get a better look at the values of each variable in our data set. The mean, max and min values of our variables helps give a deeper understanding of the dataset before creating any models.

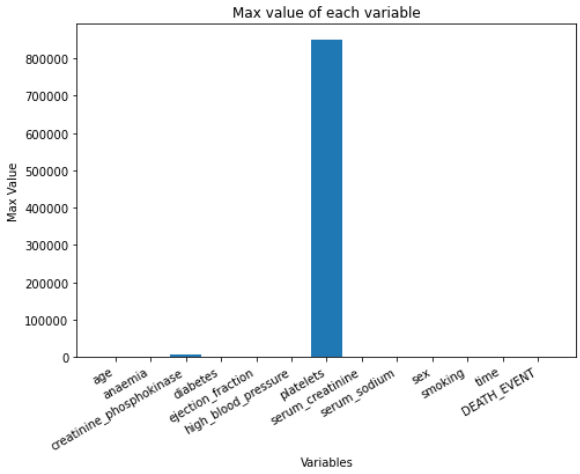

Figure 1: Bar-Chart of the Max value of each variable in the data set

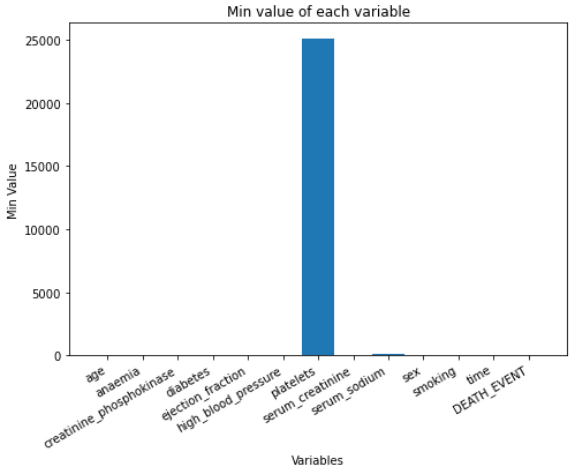

Figure 2: Bar-Chart of the Min value of each variable in the data set

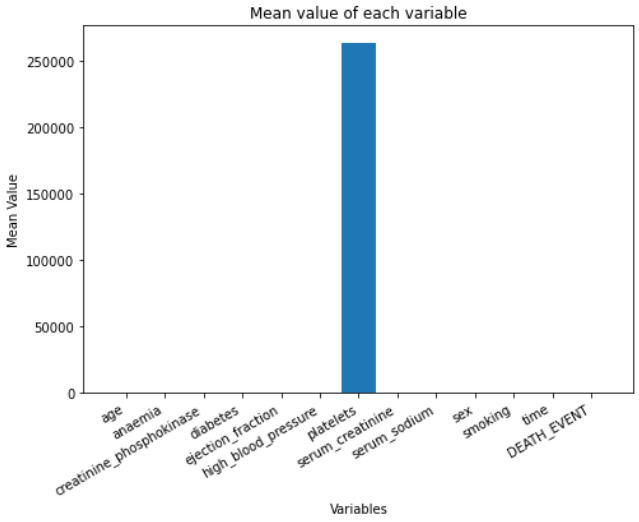

Figure 3: Bar-Chart of the Mean value of each variable in the data set

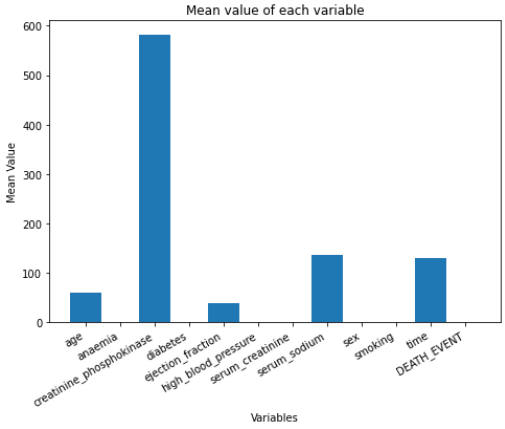

Figure 4: Bar-Chart of the Mean value of each variable in the data excluding “platelets”

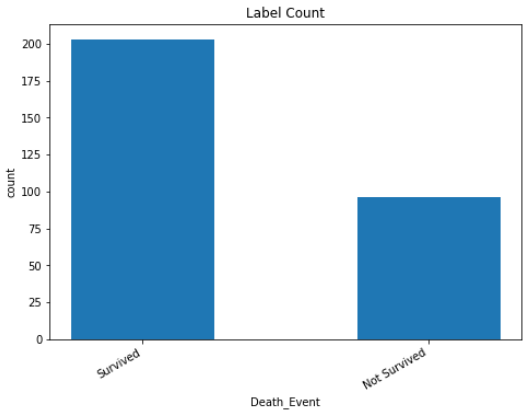

Figure 5: Label Count of the Survived vs Not Survived

From Figure 1, 2, 3 we can see that "Platelets" & "Serum Sodium" have significantly higher values than the rest of the variables. These are a perfect indicator of the need for feature's scaling. Figure 4 helps visualize the ranges of data of the remaining variables. Due to a big difference in values caused by "Platelets", the mean values of most of our data is not visible and that is why I created a fourth Bar-Chart without variable "Platelets". Figure 4 shows that half of our data is between 0-1 and the rest oscillates between 50 and 100 except for creatinine phosphokinase laying in the 600 range. From Figure 5 I can visualize potential skewness in the data. In our case we can appreciate a slight inclination towards surviving. After this profound data analysis, finding the right classifier model that can predict survival or death given unseen inputs (clinical features) is the machine learning problem we wish solving.

Training & Validation Strategy:

The code shows two machine learning methods to classify samples as surviving or not surviving heart failure :

- Artificial Neural Network: Implements MLPClassifier which is a method obtained from scikit-learns' artificial neural network library. It is based on the multi-layer perceptron algorithm and uses backpropagation. This model utilizes clinical features and corresponding labels to train and predict between 2 classes (surviving or not surviving). For the model's activation function I chose the logistic function since we are dealing with binary classification.

  Due to high varying values in features, a pipeline was created that first scales the data using StandardScaler from scikit-learns's preprocessing library and then calls the classifier.

- Support Vector Machine: uses SVC which is a method obtained from scikit-learns' support vector machine library. It is based on maximizing the classification margins. I chose the kernel type to be 'rbf' and default values for hyperparameters C & gamma. Due to highly varying values in our features, I created a pipeline that first scales the data using MinMaxScaler from scikit-learns's preprocessing library and then calls the SVC.

After creating the pipeline, I used GridSearchCV from scikit-learns' model selection package, to perform hyperparameter tunning. I gave a range of values for gamma and C to avoid long time searching and then used .best_estimator_ to select the model with the best parameters based on the model's score.

To validate the models, I created a systematic table that feeds off from a custom function called eval_model. The table displays the name of the model along with its training and test accuracy. With the help of .score, eval_model evaluates the pipelines of both models and obtains its training & test accuracy. Since accuracy is not enough to compare our binary classification models, I created another table using scikit-learn's metrics package to look at: precision, recall, f1-score, and ROC AUC score. It is important to know the number of weights of each model to get a better look at the complexity of each method.

All the metrics mentioned in this section along with the complexity of the models will help me decide which model will be best for this project.

Results & Discussion:

- Results ANN

FITTING REPORT		
Metric	Training	Test
accuracy	0.87	0.80
balanced accuracy	0.83	0.77
precision	0.80	0.93
recall	0.75	0.56
F1 score	0.77	0.70
ROC AUC score	0.83	0.77

Figure 6: ANN Metrics Table

	precision	recall	f1-score	support
0	0.76	0.97	0.85	35
1	0.93	0.56	0.70	25
accuracy			0.80	60
macro avg	0.84	0.77	0.78	60
weighted avg	0.83	0.80	0.79	60

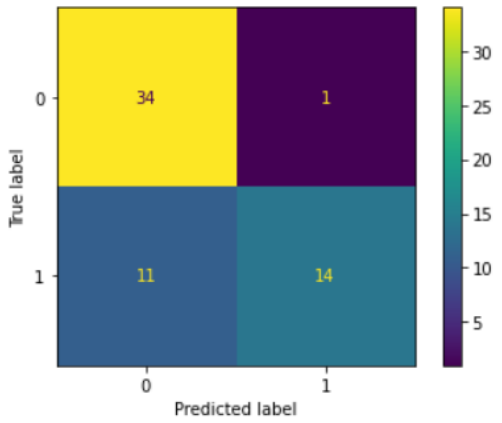

Figure 7: ANN Confusion Matrix & Classification Report

- Results SVM

FITTING REPORT		
Metric	Training	Test
accuracy	0.86	0.83
balanced accuracy	0.82	0.81
precision	0.79	0.94
recall	0.73	0.64
F1 score	0.76	0.76
ROC AUC score	0.82	0.81

Figure 8: SVM Metrics Table

	precision	recall	f1-score	support
0	0.79	0.97	0.87	35
1	0.94	0.64	0.76	25
accuracy			0.83	60
macro avg	0.87	0.81	0.82	60
weighted avg	0.85	0.83	0.83	60

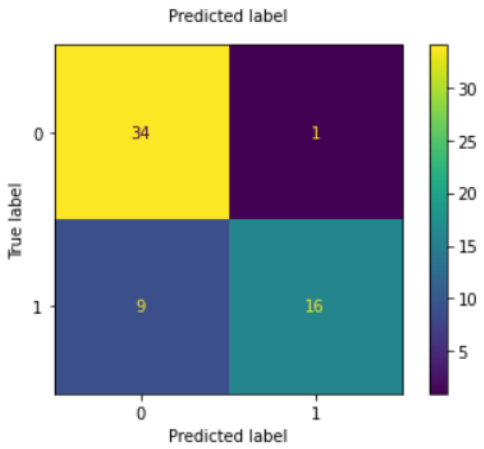

Figure 9: SVM Confusion Matrix & Classification Report

After trying different configurations, the final parameters chosen for the ANN were: logistic activation function, solver='sgd', 100 neurons in hidden layer, batch size of 25, alpha=0.01, 500 iterations and random state of 0.

The final hyperparameters chosen for the SVM model after performing hyperparameter tunning were C: 100.0 & gamma: 0.01

One of the first things to do when discussing the results of the models is to know if overfitting takes places in any of the models. To tell if a model overfits we compare the training and test accuracy. Although we typically expect training accuracy to be more than testing accuracy, if the test accuracy is significantly large in comparison with training accuracy, then we could say the model overfits. If we look at Fig 6 & 8, we can clearly see that training accuracy is higher than testing, consequently none of the models' present signs of overfitting, which is a great signal of a good, learned relationship.

The support vector machine model has far less weights than the ANN, meaning SVM is less complex. On the other hand, the ANN model has better training results than the Support Vector Machine model: higher accuracy, better accuracy for unbalanced sets, more precise, higher recall, higher F1 score and ROC AUC score. ANN slightly outperforms in all training metrics.

For this test set it seems that SVM's test metrics show better results than ANN however we may have gotten lucky with the test data.

Models show 87-86% training accuracy and 83-80% test accuracy among other scores. Considering that we were given about 300 samples, and the difficulty for medical personnel to predict heart failure deaths, both models show a good, learned relationship.

It is important to understand our problem and the context behind this project to properly select which model is best for this application. The goal consists of predicting death by heart failure given patient's clinical data. We can deduce that a mistake predicting death when it was true for surviving, is not as terrible as indicating the patient will survive and end up dying. Meaning that

our model selection should consider some wrong calls more important than others. Using the Confusion Matrices shown above, we should penalize more when predicted No Death it was True for Death (ANN= 11, SVM= 9). And we should penalize less when predicted Death and end up True for No Death. SVM's ratio of how many times the model correctly classifies members of class (No Death) is 0.79, while ANN's is 0.75.

Considering both models have almost the same accuracy 0.86 & 0.87 and around 0.01 difference in the rest of the presented metrics. Knowing that SVM model is simpler than ANN and taking into consideration the circumstances mentioned above regarding the application of this model for our Project: I believe the Support Vector Machine Classifier Model is the best candidate for our problem.

Summary: summary of main findings

During this project I used real patient's medical records to predict the mortality in patients that have previously suffered heart failures and other heart complications.

In the data provided was clearly noticeable that features like "platelets" or "creatinine" show significantly higher values (mean=250,000) all throughout the statistics in comparison with other features such as sex or smoking (mean= 0.65, mean= 0.32). That means that before applying any models, the data had to be scaled by using StandardScaler() or MinMaxScaler() and implementing a pipeline.

The two models that presented the best results were:

- ANN: MLPClassifier, logistic activation function, solver='sgd', 100 neurons in hidden layer, batch size of 25, alpha=0.01, 500 iterations and random state of 0.
- SVM Classification model: SVC(kernel='rbf'), after performing hyperparameter tunning C: 100.0 & gamma: 0.01.

Models show 87-86% training accuracy and 83-80% test accuracy among other scores. Considering that we were given about 300 samples both models show a good, learned relationship.

The goal consists of predicting death by heart failure given patient's clinical data. We can deduce that a mistake predicting death when it was true for surviving, is not as terrible as indicating the patient will survive and end up dying. We must penalize more when predicted No Death and it was True for Death (ANN= 11, SVM= 9). SVM's ratio of how many times the model correctly classifies members of class (No Death) is 0.79, while ANN's is 0.75.

SVM model is also less complex than ANN. Taking all into consideration, the Support Vector Machine Classifier Model is the best method to predict mortality in patients with heart problems in the past.