Cedric Johnson II
Jeffrey Parmet
Rowan Wiley
Dev Patel

Cardiovascular Failure

Introduction

This project investigates the Heart Failure Prediction dataset from Kaggle and evaluates the ability of the k-Nearest Neighbors, Multiple Linear Regression, Random Forests, and Learning Vector Quantization models to predict heart failure outcomes. This report introduces the dataset and details, the data analysis performed, and the resulting figures, results, interpretations, conclusions, and potential for improvement for each model.

The Dataset

The Heart Failure Prediction dataset consists of twelve clinical factors that attempt to explain and predict mortality as a result of heart failure. These predictors include:

Age
Anaemia, the decrease of red blood cells or hemoglobin
Creatinine Phosphokinase, the level of the CPK enzyme in the blood (mcg/L),
Diabetes
Ejection Fraction, the percentage of blood leaving the heart at each contraction
High Blood Pressure
Platelets
Serum Creatinine
Serum Sodium
Sex
Smoking.

The Anaemia, Diabetes, High Blood Pressure, Sex, and Smoking factors are represented using indicator variables where the value of 0 indicates the absence of this variable, while 1 indicates this factor is present for a particular patient.

The outcome, or response, variables for this dataset are Time and Death Event, as this dataset is a time-to-event dataset; the time variable describes the follow-up period for the Death Event result, which is either value 0 or 1. The value 0 represents censorship or no death, while 1 indicates mortality as a result of heart failure. The Death Event variable's categorical nature leads to its representation as an indicator variable with two values, meaning this dataset represents an example of binary classification.

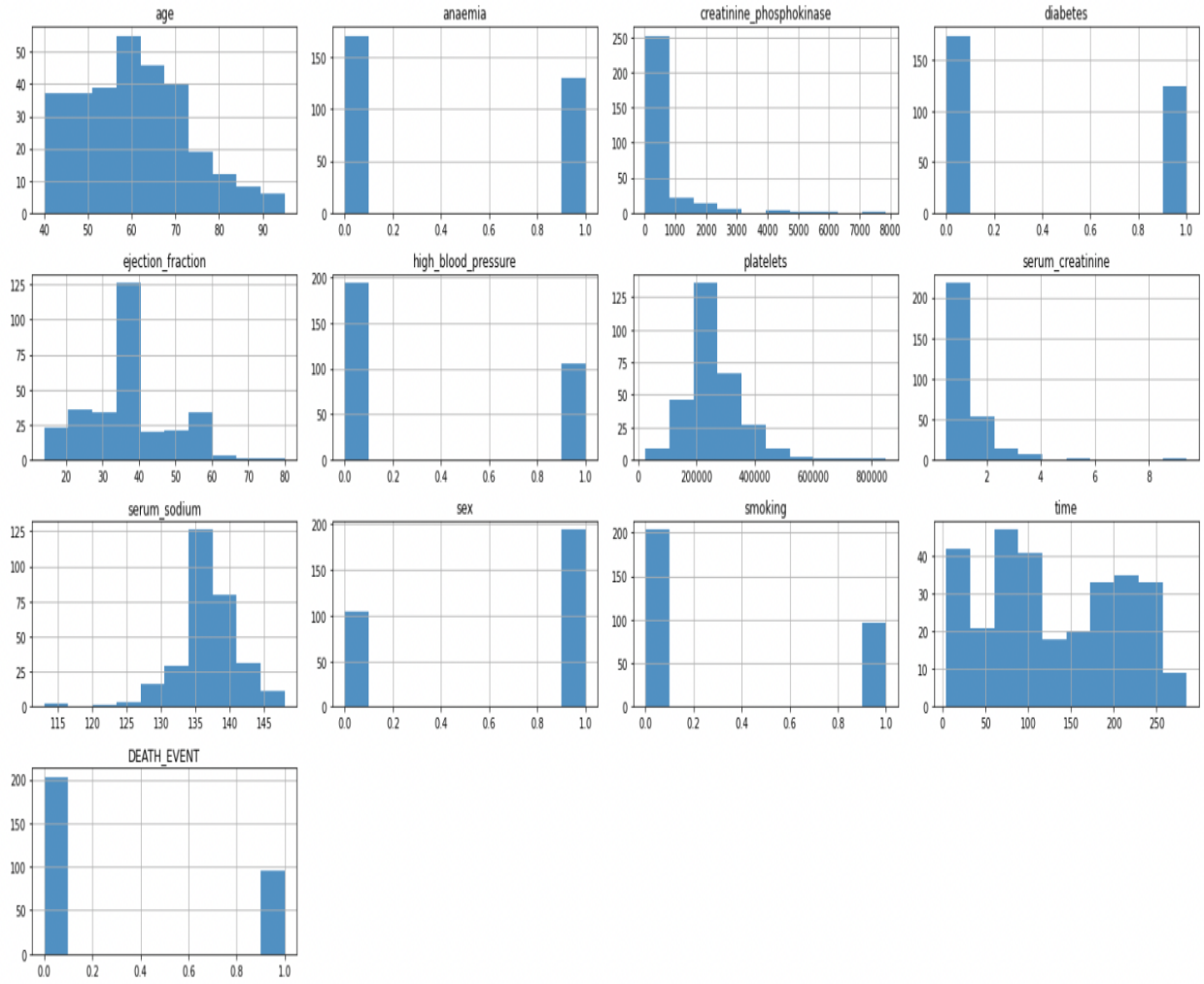

Figure 1: The distributions for each of this dataset's variables displayed using histograms.

KNN

Process

- Split the data into training and testing sets using sklearn's train_test_split function
  - Repeat this step with four total random states to get a variety of results
- Select a k-value
  - Use 17, the closest odd integer to √n
    - n = 299
- Perform model and determine measures of performance for each random state
  - Determine precision
  - Determine recall
  - Determine f1-score
  - Determine support
  - Determine confusion matrix

Results

Random State = 1

	precision	recall	f1-score	support
0	0.71	0.92	0.80	64
1	0.29	0.08	0.12	26
accuracy			0.68	90
macro avg	0.50	0.50	0.46	90
weighted avg	0.59	0.68	0.61	90

```
[[59  5]
 [24  2]]
```

Random State = 2

	precision	recall	f1-score	support
0	0.71	0.86	0.78	66
1	0.10	0.04	0.06	24
accuracy			0.64	90
macro avg	0.41	0.45	0.42	90
weighted avg	0.55	0.64	0.59	90

```
[[57  9]
 [23  1]]
```

Random State = 3

	precision	recall	f1-score	support
0	0.67	0.95	0.78	61
1	0.00	0.00	0.00	29
accuracy			0.64	90
macro avg	0.33	0.48	0.39	90
weighted avg	0.45	0.64	0.53	90

```
[[58  3]
 [29  0]]
```

Random State = 4

	precision	recall	f1-score	support
0	0.72	0.95	0.82	66
1	0.00	0.00	0.00	24
accuracy			0.70	90
macro avg	0.36	0.48	0.41	90
weighted avg	0.53	0.70	0.60	90

```
[[63  3]
 [24  0]]
```

Figure 2: KNN results

For Death Event = 0 (No death):

The precision was between 0.67 and 0.72.

The recall was between 0.86 and 0.95.

The f1-score was between 0.78 and 0.82.

The accuracy was between 0.64 and 0.70.

For Death Event = 1 (Death):

The precision was between 0.00 and 0.29.

The recall was between 0.00 and 0.08.

The f1-score was between 0.00 and 0.12.

The accuracy was between 0.64 and 0.70.

Figure 3: KNN results summarized

Observations

- The model was very good at predicting positives, correctly identifying 92.22% of true positives (237/257).
- However, it struggled with negatives, correctly identifying only 2.91% of true negatives (3/103).
- I believe this is because we used a dataset with 203 entries where death event was 0 (no death) compared to 96 entries where death event was 1 (death). This imbalanced dataset may have affected the model.
- With a k-value of 17, odds are that there will be more no death neighbors than death neighbors, just based on sheer numbers.
- This is supported by the fact that the vast majority of the errors made by the model (100/120) were false positives rather than false negatives.
- The model had an accuracy of 66.5% on average, but if it just guessed no death for every individual it would have had an accuracy of 67.89%.

Figure 4: KNN observations

We learned that data imbalance can have a significant impact on the performance of a KNN model. This makes sense, because any system where various entries "vote" on a winner will be vulnerable to such unbalanced data. This will help us better determine when it is and is not appropriate to use a KNN model. Additionally, we learned that if we want to use a KNN model, we need to make sure that the data is balanced and take steps to mitigate any imbalances that exist.

Multiple Regression

Process

- Determine number of predictors for model based on most significant correlations

```
Correlation between input features and response variable (DEATH_EVENT)

age                        0.253729
anaemia                    0.066270
creatinine_phosphokinase   0.062728
diabetes                  -0.001943
ejection_fraction         -0.268603
high_blood_pressure        0.079351
platelets                 -0.049139
serum_creatinine           0.294278
serum_sodium              -0.195204
sex                       -0.004316
smoking                   -0.012623
time                      -0.526964
DEATH_EVENT                1.000000
Name: DEATH_EVENT, dtype: float64
```

	age	ejection_fraction	serum_creatinine	serum_sodium	DEATH_EVENT
age	1.000000	0.060098	0.159187	-0.045966	0.253729
ejection_fraction	0.060098	1.000000	-0.011302	0.175902	-0.268603
serum_creatinine	0.159187	-0.011302	1.000000	-0.189095	0.294278
serum_sodium	-0.045966	0.175902	-0.189095	1.000000	-0.195204
DEATH_EVENT	0.253729	-0.268603	0.294278	-0.195204	1.000000

Figure 5: MLR observations

  - Model predictor set: Age, ejection fraction, serum creatinine, and serum sodium
  - Response: Death Event
- Split dataset into training and testing sets using sklearn's train_test_split function
  - 70/30 train:test ratio
  - Random Seed: 0

- Perform model
    - Determine intercept
    - Determine coefficients for each of the predictors
    - Determine mean squared error
    - Determine R^2
    - Determine R
    - Determine overall model equation

Results

```
Intercept:
 1.6059180286254089
Coefficients:
 [ 0.01023241 -0.0129132   0.07927147 -0.01118114]
Mean squared error: 0.19
R-Squared: 0.109517
R: 0.33093
```

Final Model Equation

DEATH_EVENT = 1.60592 + (0.01023)*Age +
(-0.01291)*EjectionFraction +
(0.07927)*SerumCreatinine +
(-0.01118)*SerumSodium

Figure 6: MLR results

Observations

Multiple Regression models determine relationships between multiple predictors and a response variable. The intended use of this model was to investigate the relationship between the response variable, death event, and the age, ejection fraction, serum creatinine, and serum sodium predictors. However, the use of multiple linear regression for this dataset does not appear to be effective, as indicated by the R^2 and R values. The R^2 value expresses that only 11% of the variance in the likelihood of a heart failure death event is explained by the use of this predictor set consisting of age, ejection fraction, serum creatinine, and serum sodium. This suggests that the predictor set could include excess factors that do not contribute to our model or the set as a whole is not effective in explaining the likelihood of heart failure. This is further supported by the R value obtained, 0.33093. This indicates that there is a relatively weak relationship between our predictor set and the outcome variable. Ultimately, this model does not serve as a great predictor of heart failure outcomes.

The use of regression with this dataset can be improved by performing logistic regression instead of multiple linear regression. Multiple linear regression models make assumptions that are invalid within the context of this dataset, particularly the assumption that the response variable is quantitative and continuous; the DEATH_EVENT categorical variable is represented by an indicator variable in this model and cannot be considered continuous, as the values are either 0 or 1. This dataset represents an example of binary classification. Logistic regression enables the resulting equation of the model to define probabilities for each of the two classes of the categorical response.

Random Forest

Process

1. Split the data into training and testing using train_test_split() with 2 random states.
   - ```
     x, y = df.drop(["DEATH_EVENT"],axis = 1), df["DEATH_EVENT"].values
     x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 2)
     ```

2. Create a RandomForestClassifier Model with 1 random state and a max_depth of 5.
   - ```
     model = RandomForestClassifier(max_features = 0.5, max_depth = 5, random_state = 1)
     ```

3. Perform model and determine measures of performance.

Results

	precision	recall	f1-score	support
0	0.84	0.84	0.84	43
1	0.59	0.59	0.59	17
accuracy			0.77	60
macro avg	0.71	0.71	0.71	60
weighted avg	0.77	0.77	0.77	60

Figure 7: RFM results

Observations

A Random Forest Model is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. We chose to use the Random Forest Model because the pros of Random Forest are that it is robust to outliers, works well with non-linear data, lowers the risk of overfitting, and runs efficiently on a large dataset. Especially since this dataset has 12 features, we wanted to lower the risk of overfitting. The model was pretty precise when determining No Death with 84% and not very precise when determining Death with 59%. This model had an accuracy of 77% which is not the best but not the worst. This model is not a strong model compared to the LVQ model, due to it being not very accurate.

Least Vectors Quantization

Process

1. Imported scikit LVQ package
2. Initialized dataset using pandas
    a. Dropped time variable and split off death_event as response
    b. Split into training and testing
    c. Scaled training data using StandardScaler
3. Initialized model
    a. Activation function, swish
    b. Distance type, squared euclidean
    c. Beta, 2
    d. Steepest gradient descent
    e. Maximum runs, 20
    f. Step size, 0.1
4. Fit the model on training data
5. Predicted the model on testing data
6. Printed classification report
7. Printed prototypes for death_event 0 and 1

Results

	precision	recall	f1-score	support
0	0.95	1.00	0.97	71
1	0.00	0.00	0.00	4
accuracy			0.95	75
macro avg	0.47	0.50	0.49	75
weighted avg	0.90	0.95	0.92	75

Figure 8: LVQ results

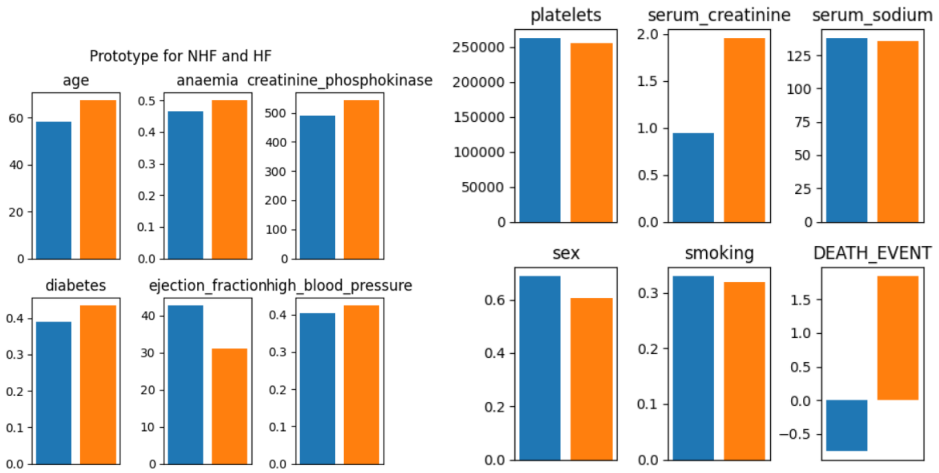

Figure 9: LVQ results continued

Observations

We learned that LVQ is a strong model related to KNN and clustering that takes advantage of the neuron structure to make predictions. The model had a high f1-score of .97 that could have continued to increase given a larger dataset. We concluded that LVQ was our most effective model when it came to making predictions based on this dataset. In particular LVQ makes a good model for handing off to non-data scientists who can use prototypes with minimal training or data background. In medical datasets such as this one, doctors can look at prototypes to determine a patient's ultimate risk of heart failure without plugging their measurements into a model.