# Logistic Regression for Heart Failure

Delgado De la Paz, Alethia Jocelyn
*Tecnológico de Monterrey Campus Querétaro*
*Intelligent Systems Technologies*
Querétaro, México
A01273709@tec.mx

***Abstract -*** ***A logistic regression model by hand was developed to predict if a patient had heart failure. Logistic Regression models are defined as statistical models that describe the relationship between a dependent variable and one or multiple independent variables [1]. Heart failure can be described as the pathophysiologic state in which abnormal cardiac performance is responsible for the inability of the heart to pump blood at a rate corresponding with the requirements of the metabolizing tissues [2]. The dataset used was obtained from the UC Irvine Machine Learning Repository [3].***

***Keywords - Logistic Regression, Heart Failure, predictions, accuracy, test, train, gradient descent, parameters, hyperparameters, attributes.***

## I. INTRODUCTION

. Logistic regression models are used to study the effects of independent variables on categorical outcomes, which normally are binary problems. Such is the case for this project where the outcome is to predict the presence or absence of heart failure. A multivariable logistic regression model was used to evaluate the information from the Heart failure clinical records. The dataset contains the medical records of 299 patients who had heart failure. Each patient has the following clinical features or attributes:

1. age: of the patient.
2. anaemia: 0 if the patient did not have it, 1 if yes.
3. creatinine phosphokinase (CPK): the amount of CPK in the blood. Normal values range from (10 to 120 micrograms per liter [4].
4. diabetes: 0 if the patient did not have it, 1 if yes.
5. ejection fraction: the percentage of how much blood the left ventricle pumps out with each contraction. Normal values range from 50 to 70 percent [5].
6. high blood pressure: 0 if the patient did not have it, 1 if yes.
7. platelets: number of platelets in the blood. The normal number ranges from 150,000 to 400,000 platelets per microliter [6].
8. serum creatinine: is a waste product in the blood that comes from the muscles. The normal level for men is between 0.7 and 1.3 mg/dL, and 0.6 to 1.1 mg/dL for women [7].
9. serum sodium: the amount of sodium in the patient's blood. The reference range is from 135 to 147 mmol/L [8].
10. sex: 0 if the patient is female, 1 if the patient is male.
11. smoking: 0 if the patient does not smoke, 1 if yes.

## II. OBJECTIVE

To create a Logistic Regression model able to predict if a patient had a death event and test the accuracy of the model.

## III. DATASET

To use the dataset for the logistic regression model first it had to be cleaned.

The first 10 samples can be seen in the Annex 1.

First the empty values in the data set were substituted with the average of the respective column.

```
for i in df.columns:
    df[i].fillna(df[i].mean(), inplace = True)
```

Then, the dataset was splitted into a training and a testing set. The test set was defined to be 20 percent of the total data, and the remaining 80 percent was used to train the model. The random_state = 0, ensures that every time the model is runned, the same split in the data is obtained.

```
X = df.iloc[:, :12]
Y = df['DEATH_EVENT']
Y = Y.values.flatten()
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, random_state = 0)
```

Then the training features were scaled with the preprocessing.StandardScaler() method from sklearn, which removes the mean and scales the variance to 1. For the training data .fit_transform is used to calculate the mean and variance of each of the features and then transform the features using the respective mean and variance. For the testing data .transform is used to transform the testing data by using the mean and variance calculated from the training data.

```
scaler = preprocessing.StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

## IV. LOGISTIC REGRESSION

To implement the logistic regression model, a class called LogisticRegression was created. The __init__() function was used to assign values to the objets learning_rate and num_iter. The learning rate is a hyperparameter often set between 0 and 1, it indicates how quickly the model adapts to the problem.

The second function inside the class is fit(self, X, Y). In this function first the parameters are initialized. The weights are initialized as an array of zeros with the same shape as the number of samples in the dataset, and the bias is defined as 0.

```
self.weights = np.zeros(features)
self.bias = 0
```

To update the parameters, gradient descent was used with a for loop that iterates the number of times specified when calling the function. Inside the for loop, the linear_model variable calculates the hypothesis function:

$$y = weights * X + bias$$

```
linear_model = np.dot(X, self.weights) + self.bias
```

The predictions are then obtained with the sigmoid function:

$$S(X) = \frac{1}{1+e^{-X}}$$

```
def sigmoid(self,X):
    return 1 / (1 + np.exp(-X))
```

The gradient descent algorithm is then applied to the parameters. The formula of the gradient descent is:

$$\theta_j = \theta_j - \frac{\alpha}{m}(hyp - y) * X_{ij}$$

Where m is number of samples, hyp is the prediction, y is the target, $\theta_j$ is the parameter, and $X_{ij}$ is the attribute. The code of the gradient descent is:

```
dW = (1 / samples) * np.dot(X.T, (Y_pred - Y))
db = (1 / samples) * np.sum(Y_pred - Y)
#UPDATE
self.weights -= self.lr * dW
self.bias -= self.lr * db
```

The prediction is calculated in another function called predict(), which receives the attributes (X) and returns the prediction. It is calculated with the hypothesis function and then the sigmoid is applied to it. If the result is above 0.5 it returns a 1, otherwise it returns a 0.

```
def predict(self, X):
    linear_model = np.dot(X, self.weights) +
 self.bias
    predictions = self.sigmoid(linear_model)
    Y_pred = [1 if i > 0.5 else 0 for i in
 predictions]
    return np.array(Y_pred)
```

To test the model, it is called with the name of the class 'LogisticRegression' with the values of the learning rate and the number of iterations.

LR = LogisticRegression(**0.01**, **1000**)

Then, to start training the model the function .fit() of the model is called with the training set as input.

LR.fit(X_train, Y_train)

To calculate the accuracy of the model, first the predictions need to be calculated. For that, the function predict() is called using the training attributes 'train_X' as input.

predictions_train = LR.predict(X_train)

Then, the accuracy was calculated with the following code:

```
def accuracy (y_true, y_pred):
    accuracy = np.sum(y_true == y_pred) /
 len(y_true)
    return accuracy
```

## V. RESULTS

After some trials, the more efficient learning rate for the model was set to 0.0065 and the model was runned for 1000 times. With those hyperparameters, the results obtained for the training and testing set were:

Training set

```
LR = LogisticRegression(0.0065, 1000)
LR.fit(X_train, Y_train)

predictions_train = LR.predict(X_train)

print("Accuracy: ", accuracy(Y_train, predictions_train))
```

```
Accuracy:  0.8577405857740585
```

With the training data, an accuracy of 0.8577 was obtained.

Testing set

```
predictions_test = LR.predict(X_test)

print("Accuracy: ", accuracy(Y_test, predictions_test))
```

```
Accuracy:  0.8166666666666667
```

With the testing data an accuracy of 0.8166 was obtained.

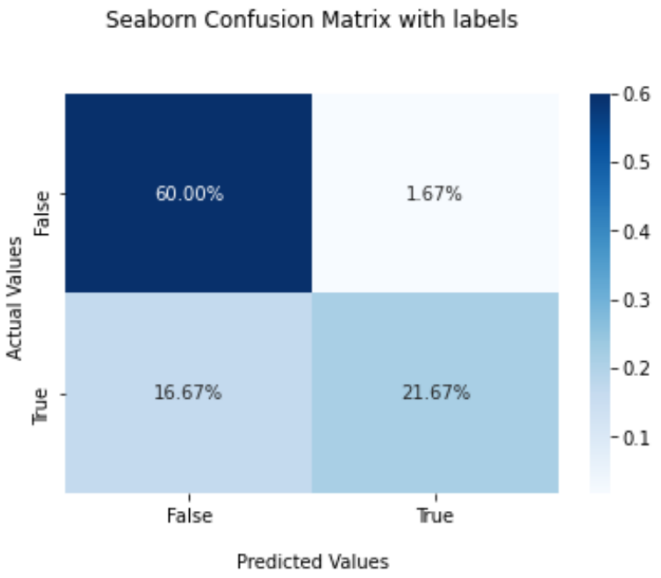


Fig. 1 Confusion matrix of the performance of the model with the testing data

## VI. DISCUSSION AND ANALYSIS OF RESULTS

The accuracy is the percentage of correct predictions made for the test data. From figure 1 the results of the model show that the amount of true negatives predicted is 60 percent, and the true positives predicted are 21.67 percent of all the predictions. This gives the 81.67 percent of accuracy obtained for the test data.

The precision of a model is the fraction of true positives among all the predictions. And is calculated as:

$$precision = \frac{TP}{TP + FP}$$

For this model, the precision is 92.84 percent.

## VII. CONCLUSION

Logistic regression models can be used as a tool to help predict if a patient has an illness or if there is a probability to develop one. As shown in this document, the accuracy and precision of these models can be somewhat admissible. But since the problem was solved with just a logistic regression, it is not recommended to use it to diagnose patients, but as a diagnostic tool. A better performance could be obtained by using a neural network. This would give a higher accuracy for each prediction. But a bigger dataset could also be useful to improve this model, so that the testing and training set have more samples to learn from.

## VIII. REFERENCES


[1] Nick, T.G., Campbell, K.M. (2007). Logistic Regression. In: Ambrosius, W.T. (eds) Topics in Biostatistics. Methods in Molecular Biology™, vol 404. Humana Press. https://doi.org/10.1007/978-1-59745-530-5_14

[2] Braunwald E. Heart failure on overview. In: Fishman AP, ed. Heart failure. New York: McGraw-Hill, 1978.

[3] Heart failure clinical records. (2020). UCI Machine Learning Repository.

[4] *Creatine phosphokinase test*. (n.d.). Mount Sinai Health System. https://www.mountsinai.org/health-library/tests/creatine-phosphokinase-test

[5] *Ejection Fraction Heart Failure Measurement*. (2022, March 30). Www.Heart.Org. https://www.heart.org/en/health-topics/heart-failure/diagnosing-heart-failure/ejection-fraction-heart-failure-measurement

[6] *Platelet count*. (n.d.). Mount Sinai Health System. https://www.mountsinai.org/health-library/tests/platelet-count#:%7E:text=The%20normal%20number%20of%20platelets,value%20ranges%20may%20vary%20slightly

[7] American Kidney Fund. (2022, February 16). *Serum creatinine test*. https://www.kidneyfund.org/all-about-kidneys/tests/serum-creatinine-test

[8] Mir, F. (2021, December 27). *Serum Sodium: Reference Range, Interpretation, Collection and Panels*. Medscape. https://emedicine.medscape.com/article/2099065-overview

# IX. ANNEXES

**Annex 1:** Heart failure clinical records dataset (first 10 samples)

| age | anaemia | creatinine phosphokinase | diabetes | ejection fraction | high blood pressure | platelets | serum creatinine | serum sodium | sex | smoking | time | Death event |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 75 | 0 | 582 | 0 | 20 | 1 | 265000 | 1.9 | 130 | 1 | 0 | 4 | 1 |
| 55 | 0 | 7861 | 0 | 38 | 0 | 263358.03 | 1.1 | 136 | 1 | 0 | 6 | 1 |
| 65 | 0 | 146 | 0 | 20 | 0 | 162000 | 1.3 | 129 | 1 | 1 | 7 | 1 |
| 50 | 1 | 111 | 0 | 20 | 0 | 210000 | 1.9 | 137 | 1 | 0 | 7 | 1 |
| 65 | 1 | 160 | 1 | 20 | 0 | 327000 | 2.7 | 116 | 0 | 0 | 8 | 1 |
| 90 | 1 | 47 | 0 | 40 | 1 | 204000 | 2.1 | 132 | 1 | 1 | 8 | 1 |
| 75 | 1 | 246 | 0 | 15 | 0 | 127000 | 1.2 | 137 | 1 | 0 | 10 | 1 |
| 60 | 1 | 315 | 1 | 60 | 0 | 454000 | 1.1 | 131 | 1 | 1 | 10 | 1 |
| 65 | 0 | 157 | 0 | 65 | 0 | 263358.03 | 1.5 | 138 | 0 | 0 | 10 | 1 |
| 80 | 1 | 123 | 0 | 35 | 1 | 388000 | 9.4 | 133 | 1 | 1 | 10 | 1 |