Melis HARMANTEPE

MACHINE LEARNING PROJECT REPORT
Heart Failure Prediction

DATASET

We have a dataset of 299 patients and 13 attributes.We are looking at 12 different risk factors (attributes) affecting each patient's chances of passing away because of a heart failure. These 12 features that can be used to predict mortality by heart failure. The last column (13th) of the dataset is the death rate, which indicates whether a patient died because of heart failure (y=1) or not (y=0).

We can observe in the last histogram of column death_rate that there are approximately 99 people who died from heart failure and 200 people who died of other reasons. So, I can say that we have a quiet balanced dataset.

Problem statement: I want to build a classification model which estimates a person's probability of deceasing from heart failure based on all of the 12 risk factors (clinical features).

DATA PREPROCESSING & EXPLORATORY DATA ANALYSIS

I start with loading the dataset into a pandas data frame in order to visualize the data. I chose to visualize the age and diabetes columns using seaborn pairplot and I observe that the number of people with diabetes or not are equally distributed over the ages of the patients. We can see that most of the people in our dataset are approximately 55 years old and there are more people with diabetes than that of not having diabetes. Then when we look at the histograms we can observe that the death rate for dying because of a heart failure (y=1) is almost double the number of deaths caused by other factors (y=0).

When I look at my dataset's column's types, I see that all of them are numerical values and thus I don't have to recategorize them or change them into numerical values. When I check if there are any null values in any of my columns, I see that there are no null values, and so I don't have to remove any of my data. I conclude that I do not need to do data cleaning. Then I look at the first 5 rows of my data and I see that the values in my columns differ a lot and they vary in different ranges, so it is inevitable to normalize the data.

Once I finish visualizing my data, I load the dataset as a bumpy array in order to be able to work with numerical data and implement functions on this data. My X matrix contains the first 12 columns and my y vector contains only the last column of the dataset, which is the death_rate. I initialize the theta vector with zeros. Then I split my data into 3 subsets: training, test and validation sets. Finally I proceed to adapting the functions to my data and compute accuracy.

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 299 entries, 0 to 298
Data columns (total 13 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   age                       299 non-null    float64
 1   anaemia                   299 non-null    int64
 2   creatinine_phosphokinase  299 non-null    int64
 3   diabetes                  299 non-null    int64
 4   ejection_fraction         299 non-null    int64
 5   high_blood_pressure       299 non-null    int64
 6   platelets                 299 non-null    float64
 7   serum_creatinine          299 non-null    float64
 8   serum_sodium              299 non-null    int64
 9   sex                       299 non-null    int64
 10  smoking                   299 non-null    int64
 11  time                      299 non-null    int64
 12  DEATH_EVENT               299 non-null    int64
dtypes: float64(3), int64(10)
memory usage: 30.5 KB
```

```
Dataset :  (299, 12) (299,)
Training set :  (179, 12) (179,)
Test set :  (60, 12) (60,)
Cross validation set :  (60, 12) (60,)
Theta :  (12, 1)
```

1. LOGISTIC REGRESSION

I use a function to traverse an array of lambda values to find the best fitting value for the Logistic Regression. I find out that the best value for lambda is 0.001. I run my model with this lambda and I acquire an accuracy of 0.83. This means that 83 percent of the time, this model can classify the death rates correctly. When we compare this value with the built-in function's accuracy, which is 0.81, we can be ensured that our model works correctly and we get a higher accuracy than an already existing function.

	precision	recall	f1-score	support
0.0	0.87	0.89	0.88	203
1.0	0.76	0.71	0.73	96
accuracy			0.83	299
macro avg	0.81	0.80	0.80	299
weighted avg	0.83	0.83	0.83	299

```
#ROC
roc_auc_score(y, y_pred)
```

```
0.7999794745484401
```

5.1 Confusion Matrix

```
cm_LR = confusion_matrix(y, y_pred)
print(cm_LR)
```

```
[[181  22]
 [ 28  68]]
```

Support shows us the number of occurrence of the given class in our dataset. We have 96 patients who died from heart failure and 203 who died from other reasons.
Precision is the ability of a classifier not to label an instance positive that is actually negative. Given that a model that produces no false positives has a precision of 1.0, this model gives a precision of 0.76 is not bad.

A model that produces no false negatives has a recall of 1.0. In our case, it's 0.71. By combining precision and recall, a perfect model achieves an F1 score of 1.0. In our case, we have an F1 score of 0.73. This is not very important when interpreting the global accuracy of the model but when we compare different machine learning models.

The compute the AUC (area under curve) and we get a result of 0.80. We know that the higher the AUC, the better the performance of the model at distinguishing between the positive and negative classes. So, this is another metric that we will use at the end of our model analysis as well as the F1 score, in order to choose the best model for our dataset.

When we look at the confusion matrix, we can observe that 181 of the total number of 299 patients were correctly classified as class 0 (y=0, death of other reason) and 68 of the 299 patients were correctly classified as 1 (y=1, death because of a heart failure).

Then, I experiment with Regularized Logistic Regression and compare the results in order to see if my dataset needs to be regularized. Looking at the Regularized Logistic Regression accuracy which is 0.53, I confirm that my dataset is very well fitted with the Logistic Regression model and there is no need for regularization.

Finally, I proceed to running the model and compare the test and training set accuracies. I get an accuracy of 0.38 on the test set and 0.82 on the training set. Since the testing accuracy is lower than the training accuracy, this indicates that there are meaningful differences between the kind of data I trained the model on and the testing data I am providing for evaluation.

Let's note that this model takes approximately 0.07 seconds to run.

2. SUPPORT VECTOR MACHINE

In this model, we first start by finding the best values of C and Sigma that fits our dataset the best. To do this, I apply the K-fold cross validation method in which we fit and train the training data and evaluate with the validation data. After finding these values we fit and train our main dataset (X and y) to get predictions. I first do this in the "rbf" type of SVM (which is used for non-linear data) and then I try the same fitting, and predicting in the "linear" kernel of SVM. I first compare the training accuracies and the execution times of these two models and the linear kernel seems to be more accurate with 0.82 of accuracy even though it takes quite a longtime to execute (51 seconds). I continue to doing the metrics with the linear model as it fits my model better than the rbf.

```
Training accuracy = 81.94%
Test set accuracy = 25.00%
Execution time is: 51.198486577 seconds
```

	precision	recall	f1-score	support
0.0	0.76	0.95	0.84	39
1.0	0.82	0.43	0.56	21
accuracy			0.77	60
macro avg	0.79	0.69	0.70	60
weighted avg	0.78	0.77	0.74	60

```
#ROC
roc_auc_score(y_test, y_pred_svc)
```

```
0.6886446886446886
```

The AUC value is 0.69.
Looking at the classification report, I can observe that the accuracy of this model is 0.82 (but the built-in function accuracy is 0.77, lower than the manual model and that is what we want), the F1 score is 0.56 which is not very appealing, precision is 0.82, and the recall is 0.43. Since the difference between precision and recall is quite high, we must take into account the F1 score.

```
[[37  2]
 [12  9]]
```

The confusion matrix tells us that 37 of 299 people were correctly classified as class 0 and 9 of 299 people were classified correctly as class 1.

3. NEURAL NETWORKS

In the neural networks model I start with traversing the array of possible lambda values and fit the training set. Then according to the maximum accuracy score, I choose the lambda value that fits my dataset the best. Then I proceed to building a model with 12 input layers and 5 hidden layers. Once I run this model, I get a training accuracy of 0.68, however the test accuracy is very low with 0.15. This model takes approximately 0.14 seconds to run.

```
Training accuracy = 68.56%
Test set accuracy = 15.00%
Execution time is: 0.14908029699995495 seconds
```

```
              precision    recall  f1-score   support

         0.0       0.65      1.00      0.79        39
         1.0       0.00      0.00      0.00        21

    accuracy                           0.65        60
   macro avg       0.33      0.50      0.39        60
weighted avg       0.42      0.65      0.51        60
```

```
#ROC
roc_auc_score(y_test, y_pred_nn)
```

```
0.5
```

```
cm_NN = confusion_matrix(y_test, y_pred_nn)
print(cm_NN)
```

```
[[39  0]
 [21  0]]
```

Again, I use the built-in function to evaluate the accuracy and I get 0.65 (which is lower than the manual model and thus coherent). I get 0.0 for all of precision, recall and F1 score and this is horrible result. Even though the accuracy can reach 0.65, this is not reliable because the accuracy is very easy to compute and is a very robust metric.

The confusion matrix is also very bad with 39 true positives and 0 true negatives.

This model is by far the worst that fits our dataset.

	Logistic Regression	SVM	Neural Networks
Accuracy	0.83	0.82	0.68
F1 Score	0.73	0.56	0.0
AUC	0.80	0.68	0.5
Execution time (secs)	0.07	0.14	51

CONLUSION

In conclusion, let's start with comparing the easiest metric for each model : accuracy. Logistic regression has 0.83, SVM has 0.82 and Neural networks has 0.68 of accuracy. We can observe that the Logistic regression model has the best accuracy, whereas the worst accuracy is given by the Neural networks. When we compare the F1 scores, Logistic regression has 0.73, SVM has 0.56 and Neural networks has 0.0. Since the F1 score is one of the most important metrics when it comes to model comparison, this shows that clearly the best model is the Logistic regression, then SVM and lastly, the worst is Neural networks. Another metric which is very important in this regard is the area under the curve. For Logistic regression this value is 0.80, for SVM it is 0.68 and for Neural network it is 0.5. Again it is clear that Logistic regression is the best model. Lastly, if we compare the execution time of each model, we see that the fastest is the Logistic regression with 0.07 seconds, and the slowest is the SVM with 51 seconds (this is because the linear kernel takes so much time to run, but rbf is fast).

Overall, the Logistic regression model is by far the best model that fits this dataset in the best way and that is the most reliable model. We can also make use of the SVM model however the Neural network model is the worst model for this dataset.