Quillen Flanigan

CS 591

Dr. Atkison

12/1/2023

Assignment 2 Lab Report

1. **Introduction**

This lab serves the primary purpose of analyzing a dataset and using the gained insight to develop a predictive model for the particular dataset at hand. Specifically, we are given a dataset of medical records from patients who have suffered from heart failure. This data includes several risk factors commonly associated with heart failure, including smoking, age, diabetes, blood pressure, etc. We are then tasked with processing this data, using the comprehensive information to develop a predictive model for the likelihood of a patient suffering from a fatal heart condition. Through the use of three models, namely the Random Forest Classifier, K-Nearest Neighbors Classifier, and Decision Tree Classifier, we will work to generate a predictive tool as a method of identifying, managing, and preventing cardiovascular diseases.

1. **Initial Insight into the Clinical Dataset**

The provided dataset contains 12 common indicators often linked to an elevated risk of heart failure, and a 13th value named “DEATH\_EVENT”, which we will use our models to try and predict based on the first 12 values. The first step in this process of understanding the data was to check for any missing values, categorical variables, Boolean values, or any other blatant outliers that may interfere with the model’s ability to train from the data. The dataset did not have any missing values and consists of solely integer or float values, meaning there was no need for any Boolean-to-integer or categorical-to-numerical conversions. However, the first examination of the data indicated a pattern that could be a potential issue. The ‘time’ variable, indicating the follow-up period (in days) during which the patient’s health was monitored, was the only value that was in order throughout the entire dataset. Every variable is distributed seemingly at random with no pattern or apparent sorted order. However, the ‘time’ variable is in ascending order, beginning with the first patient at 4 days and ending with the last at 285 days. This unique ordered property could lead to bias in the future if the model were to interpret this ordering as an important property of the data. To account for this, we used the “sample(frac=1)” method to return a random shuffling of the entire data set (signified by the frac=1 expression). This means our dataset is now randomly shuffled, getting rid of any ordering within any variable that may distract the model from other variables.

After these initial steps, we move into generating various descriptive statistics on the data set to gain insight into any patterns, outliers, or potential bias within the data. After reading the dataset in with the *panda’s* library method ”read\_csv()”, we used the “describe()” method to generate various metrics on the raw data set, including the mean for each factor, maximums and minimums, frequency, and standard deviation. This allowed us to gain quick insight into any obvious patterns within each indicator, while also ensuring that there were in fact no missing values.

A screenshot of a computer

Description automatically generated

Figure 1. Descriptive Statistics of each health indicator within the dataset

* 1. **Standardization and Preprocessing**

Fig. 1 shows each of our 13 data values and their corresponding statistics. We see that each value has a count of 299, verifying that there are no missing values. This “describe()” method also depicts several of our indicators that have both a large standard deviation and a large variance between the minimum and maximum values. For example, the *platelets, time,* and *creatinine\_phosphokinase* indicators all have a much more drastic difference between their minimum and maximum values, and their standard deviation, than the other values. This variance can often prove to be problematic for machine learning models as the model becomes “distracted” by the inconsistency and may lose the ability to effectively learn from the provided data set. When a small group of the variables within a dataset have this much more variance relative to the rest of the data, the estimators within each model might be dominated by these inconsistent features and may be unable to gather data from the rest of the variables. This could introduce potential bias within our models, harming its generalization ability and performance on future data. To reduce this variance within each data value, we use the StandardScalar() function from the *sklearn.preprocessing* library. This preprocessing tool allows us to standardize each individual value within our data set, making the dataset more closely resemble a normal distribution. By declaring a new instance of the StandardScalar() function and using its “transform()” method, we can scale our data into a more consistent set that will reduce any bias in the model towards any particular variable. We then use the “DataFrame()” function from the *pandas* library to convert our scaled data back into a pandas array-like data structure, allowing us to use the “describe()” and “hist()” functions to gain further insight into the standardized data.

A screenshot of a computer program

Description automatically generated

Figure 2. Descriptive statistics of the standardized data

Our data now looks much different, as we have completely reduced the variation within any single indicator by subtracting the mean of each value and then dividing each by the standard deviation. This transforms each value into a more compressed, normalized value so the model can better interpret information from each variable. Although we lose some interpretability of the data, the model’s performance and comprehension of the data will be dramatically improved. We can use the aforementioned “hist()” function for each dataset to generate a more visual representation of the data.

A screenshot of a graph

Description automatically generated

Figure 3. Comparison of each histogram for the raw and standardized data

Fig. 3 depicts this visualization, as the “hist()” function generates a histogram for each variable within the dataset. Importantly, the histograms for both the standardized data (left) and the raw input data (right) are very similar. Although the statistics for each dataset differ greatly, the histogram distributions are essentially the same as the relationship between each variable has not been noticeably affected. The standardization makes the variance between values much smaller, but the rate of difference remains similar, meaning the distributions of the standardized data are proportional to those of the raw data. Maintaining the proportionality is important to maintain the original properties and relationship between each variable, which could hold patterns that may influence the model’s predictive ability.

Now that we have gained an insight of possible red flags, shuffled the data to account for any disruptive ordering, and used standardization to reduce variance and account for any potential bias within the dataset, we are ready to instantiate our three models and train them to both the raw and scaled data. Using both datasets allows us to compare and contrast the model’s performance on each set and view the effects that the standardization has on the model’s ability to learn from intricacies within the data.

* 1. **Feature Importance Processing**

During analysis of the model’s feature importance’s, the ‘time’ variable dominated the feature importance score for the RF model, indicating a non-sensical result of feature importance (time is not more indicative than smoking, for example), most likely as a result from feature dominance. We attempted to remove the ‘time’ variable to remove the bias in the model, which helped the feature importance make more real-world sense but the led to a large reduction in performance for all three models. These findings are included in the appendix section at the end of the report.

1. **Model Implementation**

Now that our data has been processed and standardized, we move into declaring instances of each of our three models: the Random Forest Classifier, K-Nearest Neighbor Classifier, and the Decision Tree Classifier. Each of these models carry their own benefits and drawbacks relative to performance metrics, computational efficiency, complexity, simplicity, and interpretability. Firstly, the Decision Tree (DT) Classifier are popular in supervised learning working to predict a value by building a tree of decision nodes that are interpreted from the features and indicators within the data. These DTs are often simple and interpretable, computationally cheap, and generally perform well on most generalized data. However, the trees themselves can get over-complex which can lead to overfitting and heavy bias if there are dominant features within the data. Random Forest (RF) Classifiers are groups, or ensembles of Decision Trees, each of which is constructed from a sample of the training data. Lastly, the K-Nearest Neighbor (KNN) Classifier relies on using the distance between data points to predict and create labels for the data. This model often stores and uses the samples of near points to classify future data, rather than constructing a general predictive model. These KNNs are often quite simple in implementation and computation, yet often perform worse on large datasets and can often be skewed from too much variation within the dataset.

* 1. **Split the Training Data and Train the Models**

To declare our instances of these models, we use the three functions RandomForestClassifier(), KNeighborsClassifier(), and DecisionTreeClassifier(), respectively. Once this blank slate for each model is declared, we then use the “fit()” method for each model to train on the data, passing the split training data we generate from the train\_test\_split() function. Once we shuffled our dataset to remove any ordered bias, we split the set into an X set, including every column except the “DEATH\_EVENT” value, and a Y set which is the DEATH\_EVENT column. This allows us to separate the indicators the model will use to build its predictive abilities, and also have the expected values the model will be comparing its predictions to in order to gauge its performance. We pass these X and Y lists into the train\_test\_split() function, returning a training and a validation set for both X and Y. We also set the test\_size parameter within the train\_test\_split() function, which we initially set to 20% (meaning 80% of the data is for training and 20% for validation testing), and then changed to 10% to compare any performance increases from the extra 10% of data the model was able to train with. We wanted to account for any potential overfitting to the data, so 10% served to be our low bound for the validation data and 20% served as the baseline value.

* 1. **Model Predictions**

Once our data was split into training and validation sets, we pass the X\_train and Y\_train sets into the “fit()” method for each model, training each of our three classifiers on the same data. This returns three trained models that have been fitted on the patterns and intricacies within our data set. The models will then use this data in order to predict the target value based on future indicators as input. After the models have been trained, we then use the “predict()” method for each model to predict the target DEATH\_EVENT value on the reserved validation data. We pass the models the X\_test subset of the validation data into the predict() function, returning a list of predictions. This list will be a binary classification (0 or 1), indicating whether or not the model believes the indicators of each patient within the X\_test data will lead to a death (1) or survival (0). We then use the Y\_test subset of the validation data, containing the expected values, to compare to each model’s prediction for each patient. This allows us to judge the performance of each model, specifically the accuracy, precision and recall performance metrics.

1. **Model Performance Evaluation and Comparison**

To compare the performance of each model with respect to accuracy, recall and precision, we will use the imported “accuracy\_score()” and “confusion\_matrix()” functions. Because we have both the raw data and the standardized data, we chose to evaluate both sets with our models to judge the importance of the standardization. Firstly, we use the accuracy\_score() function, passing in the Y\_test list of expected values, and the list of each model’s respective predictions on the validation set. The function then returns a float value representing the percentage of correct predictions. Each iteration of running each model produced slightly different results, so we chose to run six iterations for each model, averaging the accuracy scores for each model and compiling them into a plot graph to visualize their performance. Overall, the RF model consistently performed the best out of each model, averaging an 85% accuracy score on the 10% test\_size iterations and 86.1% on the 20% iterations. The DT model was consistently the second best performing at 71% and 84%, respectively, with the KNN model being the worst performing at 66% and 63%. These accuracy scores for each of the 6 iterations for each model are displayed below in Fig. 4 and Fig. 5, representing the 10% and 20% test size splits, respectively. A graph of lines and numbers

Description automatically generated with medium confidence A graph of different colored lines

Description automatically generated

Figure 4. 10% test\_size accuracy for each model Figure 5. 20% test\_size accuracy for each model

Now that we have evaluated the accuracy scores of each model that was trained on the raw data, we now move to evaluating the impact the standardizations had on the performance of each model. We first created a list of the shuffled values using the “.values” attribute of our shuffledData list. We then split this list of values in a similar fashion to the raw data, using array splitting with our scaledX list being each of the 12 indicator variables and the scaledY list being our target value list. These values will then be placed within the train\_test\_split() function with the same respective test\_size() values as the raw data. However, this requires we format the data into non-continuous input, meaning we apply the “astype(‘float’) function to the scaledX list, setting each value as a float to ensure we do not have continuous-typed data within our splitting function as the model evaluation we use cannot accept continuous data. We then use the LabelEncoder() function to perform a similar process on the scaledY list, setting the type as “str” within the fit\_transform() method of the LabelEncoder. This again declares our expected values list as non-continuous, allowing us to then use the split data within our fit() function for the standardized data.

To be able to test the evaluation of our model on the standardized data, we use the Pipeline() function which generates an instance of our respective model that is able to accept the new data set. We create three pipelines, one for each of our models, and fit each one with the scaledX\_train and scaledY\_train data from the train\_test\_split() function. We now again have three trained models, but they are not fitted to the standardized data. This process is to ensure that we limit as much potential bias and variation within the model’s interpretation of the dataset to ensure the model can properly learn from the patterns within each variable. We then use the cross\_val\_score() function, passing our fitted pipeline for each model, along with the scaledX and scaledY data sets. This returns a list of averaged accuracy scores from 10 iterations of each model. This means the cross\_val\_score function ran the passed model 10 times (which we declared with the parameter n\_splits=10), which we then iterated through and calculated the mean of each pipeline instance. This provided us with an average of accuracy scores for all three models that were trained on the standardized data. The results from these scaled models are also displayed in Fig. 4 and Fig. 5.

The most important factor within these accuracy results is the level of variation within the accuracy scores, as we see the models that were trained on the raw data show much higher levels of inconsistency compared to the models that were trained on the standardized data. The six iterations of each model’s accuracy score are much more dispersed for the models trained on the raw data, most likely as a result of the variation and noise within this original data set. These models trained on the raw data are not able to learn from each variable as effectively as the three scaled models, which were able to view the patterns between each indicator more easily. The three original models were able to accomplish higher maximum accuracy scores, but also were responsible for the lower bounded range of the accuracy. This reflects poorly on the model’s ability to be implemented in real-world applications, as the consistency of the model, especially in the highly sensitive field of healthcare, is arguably more important than the accuracy.

* 1. **Model Parameter Adjustments**

Given this disparity in performance from the original models, we decided to alter the default parameters of each model. Firstly, we looked at the KNN model due to its performance being the worst, likely as a result of its inherent property of not developing an internal predictive model. The KNN model is often referred to as a “Lazy Learner”, meaning it performs on instance-based learning rather than forming a predictive model for future data. To adjust for this shortcoming, we attempted to change the main classification algorithm of our model from “auto” to “kd\_tree”, as we wanted to resemble the tree-like structure of the RF and DT models that seemed to be outperforming the KNN. This tree like structure seems to perform better than the instance-based lazy learning, as the indicators within the dataset often combine to give insightful clues as to what variables lead to a certain outcome within the target value. Decision Trees and Random Forests are particularly equipped to handle this “decision-based” learning. This change to the KNN algorithm then led to an increase in accuracy score of about 7%, from 66% to 73%, the current performance displayed in Fig. 4 and Fig. 5.

This concept of adjusting model parameters then spilled over into the RF and DT models, as we began tinkering with the default number of estimators and default decision-making criterion that was used in the RF and DT models, respectively. Through a process of mainly trial and error analysis, we were able to improve the accuracy scores of both tree-based models from around 79% to 85% for the RF model and 73% to 82% for the DT model. We updated the number of estimators within the RF model from 100 to 200, attempting to double this value to increase the decision-making capacity of the RF as a whole. This process yielded mostly diminishing marginal returns, as the only increase of value came from the original increase to 200 and every value increase after that (until 1000) led to marginal accuracy increases but very large drawbacks in computational efficiency. We then changed the ‘criterion’ metric for the DT model from the default of the Gini Impurity function to the entropy function. The entropy seemed to be more appropriate for this model, as it decides which decision split based on the data feature with the least amount of entropy, or randomness. Because our data had large amounts of entropy at some points, especially with the increased variance within the raw data, the entropy function was able to make up for some of this deviation and make a more accurate model, leading to about a 5% increase in both the scaled and original DT models.

* 1. **Confusion Matrix Comparison**

After analyzing the accuracy score performance of each model, we then moved onto comparing the confusion matrices for each model, allowing us to compare the models performance in precision and recall in addition to accuracy. Precision is defined as the ratio between True Positives (TP) and all positives (TP + FP), essentially asking “what proportion of positive predictions were correct?”. Recall seeks to answer the question “how many/what was the rate of actual positives that were identified correctly?”. These metrics combine with accuracy form a more comprehensive measure of performance, as we can compare the model’s ability to distinguish between more specific cases of positive and negative identifications. Using the confusion\_matrix() function, we passed each of our models trained on the original raw data to return the number of times each model registered a true positive (TP), false positive (FP), false negative (FN) and true negative (TN). We then generated each model’s confusion matrix 6 times to gain a comprehensive average of their performance and did so for both the 10% test size and 20 test size.

The overall findings, shown in Fig. 6-11, show that the same pattern of each model’s performance in accuracy holds for the confusion matrices, as the RF model seemed to perform the best, followed by the DT model, then the KNN model. Furthermore, all three models showed better, more consistent results for the 20% test size compared to the 10% test size, most likely due to less risk of overfitting and more test cases in the 20% size meaning there is a higher chance for the model to produce normalized results.

A graph of different colored lines

Description automatically generated A graph with colored lines and text

Description automatically generated

Figure 6. RF 10% confusion matrix Figure 7. RF 20% confusion matrix

A graph of different colored lines

Description automatically generated A graph of different colored lines

Description automatically generated

Figure 8. KNN 10% confusion matrix Figure 9. KNN 20% confusion matrix

A graph of different colored lines

Description automatically generatedA graph of different colored lines

Description automatically generated

Figure 10. DT 10% confusion matrix Figure 10. DT 20% confusion matrix

* 1. **Overall Comparisons**

As we can see from the above plots of each confusion matrix iteration, each of our model’s perfomance was greatly improved by the increase in test\_size, as we get a more normalized matrix in return. The plots show the values of each model’s frequency for TP, FP, FN, and TN for each iteration. The best performing models again seem to be those that have a tree-based strcture, as that structure applies very well to the decision making process needed for this feature-based classification. The desired outcome of each confusion matrix is to maximize the TP and TN values while minimizing the FP and FN values, thus creating a U or W shape within the graph. The ‘closeness’/’tightness’ of each iteration also displays the consistency for each model that is valued within this application of machine learning. The model must produce consistent results in order to be a useful tool in correctly identifying potential cardiovascular disease. With this in mind, the RF model with a 20% test size seems to be the best option in terms of both the general accuracy score and the confusion matrix, especially if trained on the standardized data to remove any distracting features for the model. This is followed by the DT model, who shares the pattern of much improved consistency from the 10% to 20% test size in the confusion matrix performance, along with the increased accuracy as a result from the standardized data. Lastly, the KNN model is the overall worst in both categories of performance, as it continuously showed elevated levels of false negatives, showing an inability to idenfity when someone is at risk for fatal heart disease. The KNN also proves to be the most inconsistent, again reflecting the importance of the tree-like decision making structure that seems to be more appropriate for this application as opposed to the lazy-learning approach of the KNN.

1. **Discussion of Real World Applications**

This application of machine learning will continue to prove to be highly important and impactful to real world scenarios. The overall field of healthcare will continue to evolve and develop a partnership with the field of machine learning, as demonstrated to be effective from this lab. Overall, the Random Forest Classifier model proved to be the most successful in both predicting the likelihood of fatal heart disease risk (true positives), correctly identifying it not to be a risk (true negatives), while also boasting the highest accuracy score among all three models. The relative success of the tree-based models, namely the RF and the Decision Tree model, indicate that the decision tree foundation is the most ready and applicable to real world scenarios that require classification based on a number of indicative features.

The fact that the RF model was able to accurately predict the correct classification 86% of the time while also proving the most effective in limiting false positives and false negatives indicates that machine learning can certainly be a valuable tool within healthcare, but also any field that requires a similar process of decision making. For now, the applicability seems to be that of an aiding tool rather than something that medical professionals can fully rely on. However, this model can serve to be very effective in identifying risk for large groups of people, something that can be handled especially well by the RF model.

**Appendix:**

RF accuracy score is: 0.8

K-neighbor accuracy score is: 0.5

Decision-Tree accuracy score is: 0.8

RF Summary

-----------------------------------------------------------------------

Feature: age -> feature importance score: 0.07800368965159074

Feature: anaemia -> feature importance score: 0.01364467076873129

Feature: creatinine\_phosphokinase -> feature importance score: 0.08142458202334082

Feature: diabetes -> feature importance score: 0.014709002320593486

Feature: ejection\_fraction -> feature importance score: 0.11316567824041705

Feature: high\_blood\_pressure -> feature importance score: 0.014545000836071003

Feature: platelets -> feature importance score: 0.07920082400663005

Feature: serum\_creatinine -> feature importance score: 0.14695242236290498

Feature: serum\_sodium -> feature importance score: 0.07743300964557591

Feature: sex -> feature importance score: 0.012780468330893465

Feature: smoking -> feature importance score: 0.015422738344741373

Feature: time -> feature importance score: 0.3527179134685098

Permutation Importances for RFModel:

Feature: age -> feature importance score: [0.02006689 0.01672241 0.01337793 0.02006689 0.02006689]

Feature: anaemia -> feature importance score: [0. 0. 0. 0. 0.]

Feature: creatinine\_phosphokinase -> feature importance score: [0.01003344 0.01003344 0.01003344 0.01337793 0.00668896]

Feature: diabetes -> feature importance score: [0. 0. 0. 0. 0.]

Feature: ejection\_fraction -> feature importance score: [0.06020067 0.04347826 0.04347826 0.05016722 0.05016722]

Feature: high\_blood\_pressure -> feature importance score: [0. 0. 0. 0. 0.]

Feature: platelets -> feature importance score: [0.02006689 0.02006689 0.01337793 0.01672241 0.01672241]

Feature: serum\_creatinine -> feature importance score: [0.06020067 0.06688963 0.05016722 0.06354515 0.04347826]

Feature: serum\_sodium -> feature importance score: [0.03010033 0.02006689 0.01003344 0.02006689 0.02341137]

Feature: sex -> feature importance score: [0. 0. 0. 0. 0.]

Feature: smoking -> feature importance score: [0.00334448 0. 0.00334448 0.00334448 0. ]

Feature: time -> feature importance score: [0.2541806 0.22073579 0.21070234 0.2541806 0.21070234]

K Summary

-----------------------------------------------------------------------

DT Summary

-----------------------------------------------------------------------

Feature: age -> feature importance score: 0.04926923784973779

Feature: anaemia -> feature importance score: 0.0

Feature: creatinine\_phosphokinase -> feature importance score: 0.05344547474262071

Feature: diabetes -> feature importance score: 0.009891284338610383

Feature: ejection\_fraction -> feature importance score: 0.09511281580920508

Feature: high\_blood\_pressure -> feature importance score: 0.0

Feature: platelets -> feature importance score: 0.06766301205360234

Feature: serum\_creatinine -> feature importance score: 0.14084738047053968

Feature: serum\_sodium -> feature importance score: 0.04046377384921474

Feature: sex -> feature importance score: 0.016022019294981295

Feature: smoking -> feature importance score: 0.0

Feature: time -> feature importance score: 0.5272850015914881

Permutation Importances for DTModel:

Feature: age -> feature importance score: [0.02006689 0.03344482 0.0367893 0.02341137 0.02675585]

Feature: anaemia -> feature importance score: [0. 0. 0. 0. 0.]

Feature: creatinine\_phosphokinase -> feature importance score: [0.0367893 0.03010033 0.02006689 0.03010033 0.04013378]

Feature: diabetes -> feature importance score: [0.00334448 0.01003344 0.01672241 0.01337793 0.00668896]

Feature: ejection\_fraction -> feature importance score: [0.06688963 0.06354515 0.11036789 0.0735786 0.08026756]

Feature: high\_blood\_pressure -> feature importance score: [0. 0. 0. 0. 0.]

Feature: platelets -> feature importance score: [0.0367893 0.03344482 0.02341137 0.03344482 0.02341137]

Feature: serum\_creatinine -> feature importance score: [0.1270903 0.13043478 0.11705686 0.11036789 0.11705686]

Feature: serum\_sodium -> feature importance score: [0.03010033 0.02006689 0.03010033 0.03010033 0.02341137]

Feature: sex -> feature importance score: [0.02341137 0.02006689 0.02006689 0.01672241 0.03010033]

Feature: smoking -> feature importance score: [0. 0. 0. 0. 0.]

Feature: time -> feature importance score: [0.2541806 0.28093645 0.27090301 0.29765886 0.24414716]

**A graph of blue and black bars

Description automatically generated with medium confidence**