Heart Disease Machine Learning Pipeline

Abdullah Khalid
Department Of Computer Science
Loughborough University
Loughborough, UK
a-khalid-22@student.lboro.ac.uk

I. Dataset:

To make a proper pipeline we need to first understand the whole dataset. Proper understanding of the dataset can help us determining if the prediction is correct or not. Lets look at all the fields and see what they all mean.

1) age: The persons age in years
2) sex: The persons sex (1 = male, 0 = female)
3) cp: Chest pain in 4 values (0: typical angina, 1: atypical angina, 2: non-anginal pain, 3: asymptomatic)
4) trestbps: Resting blood pressure
5) chol: Cholesterol measurement in mg/dl. This plays a role in heart disease but its not that important.
6) fbs: Fasting blood sugar (¿ 120 mg/dl, 1 = true; 0 = false)
7) restecg: Resting electrocardiographic measurement (0 = normal, 1 = ST-T wave abnormality, 2 = showing probable or definite left ventricular hypertrophy by Estes criteria)
8) thalach: Maximum heart rate achieved
9) exang: Exercise induced angina (1 = yes; 0 = no)
10) oldpeak: ST depression induced by exercise relative to rest
11) slope: The slope of the peak exercise ST segment (0: upsloping, 1: flat, 2: downsloping)
12) ca: The number of major vessels (0-3)
13) thal: A blood disorder called thalassemia (1 = normal; 2 = fixed defect; 3 = reversable defect, 0 could be missing/error data)
14) target label: Heart disease (0 = no, 1 = yes)

In the dataset we can see some categorical values and some are continuous values like chol etc

A. Categorical Values:

All the features whose values are categorised as in gender can either be male or female, are called categorical values. Our data set has 8 categorical values excluding target because later well drop the target feature.

1) cp
2) thal
3) slope
4) sex
5) fbs
6) exang
7) ca
8) restecg

These all are the categorical values as they rate their intensity or values in categories. Categorical Values divides the data set into several different categories such as people who have a disease and people who don't etc.

B. Continuous Values:

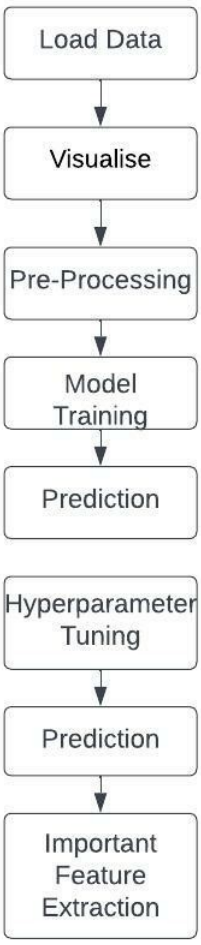
All those features whose values are continuous rather than being categorised are known as continuous values. Such as Chol or Cholesterol is a continuous value because it's not 0 or 1 and each instance has its own separate or in some cases unique value. In our data set we have a few continuous values.

1) Age
2) Trestbps
3) Chol
4) Thalach
5) Oldpeak

Later on, we'll perform different pre-processing on Continuous Values and Categorial Values.

II. Methods:

A. Machine Learning Pipeline:



B. Data Pre-Processing:

In the previous section we talked about how our data set has 2 kinds of values. Continuous values and categorical values. Now we'll talk about how we treated these two kinds of values.

1) Processing Categorical Values:: So, for the categorical values we created dummy values for each of the feature. To create these dummy values, we used pd.get_dummies. What it did was that it converted all the categorical values into separate columns. For example, sex can be either 0 or 1 so we converted sex column to sex_0 and sex_1. After performing pre-processing on the data set, it looked something like this.

```
In [17]: Dum_CP = pd.get_dummies(df['cp'], prefix = "cp")
         Dum_Thal = pd.get_dummies(df['thal'], prefix = "thal")
         Dum_Slope = pd.get_dummies(df['slope'], prefix = "slope")
         Dum_Sex = pd.get_dummies(df['sex'], prefix = "sex")
         Dum_Fbs = pd.get_dummies(df['fbs'], prefix = "fbs")
         Dum_Exang = pd.get_dummies(df['exang'], prefix = "exang")
         Dum_ca = pd.get_dummies(df['ca'], prefix = "ca")
         Dum_Restecg = pd.get_dummies(df['restecg'], prefix = "restecg")
         frames = [df, Dum_CP, Dum_Thal, Dum_Slope,Dum_Sex,Dum_Fbs,Dum_Exang,Dum_ca,Dum_Restecg]
         df = pd.concat(frames, axis = 1)
         df = df.drop(columns = ['cp', 'thal', 'slope','sex','fbs','exang','ca','restecg'])
         df.head()
```

Out[17]:

	age	trestbps	chol	thalach	oldpeak	target	cp_0	cp_1	cp_2	cp_3	...	exang_0	exang_1	ca_0	ca_1	ca_2	ca_3	ca_4	restecg_0	restecg_1	restecg_
0	63	145	233	150	2.3	1	0	0	0	1	...	1	0	1	0	0	0	0	1	0	
1	37	130	250	187	3.5	1	0	0	1	0	...	1	0	1	0	0	0	0	0	1	
2	41	130	204	172	1.4	1	0	1	0	0	...	1	0	1	0	0	0	0	1	0	
3	56	120	236	178	0.8	1	0	1	0	0	...	1	0	1	0	0	0	0	0	1	
4	57	120	354	163	0.6	1	1	0	0	0	...	0	1	1	0	0	0	0	0	1	

5 rows × 31 columns

One may ask why we converted our categorial values to dummy variables. So, the answer to this is that Dummy variables transform categorical data into numerical data which in theory helps us to create a better fit for the data.

2) Processing Continuous Values:: So, for the continuous values, we'll use Standard Scalar. The idea behind StandardScaler is that it will transform data such that its distribution will have a mean value 0 and standard deviation of 1. Which means each column/feature will have μ= 0 and σ = 1. (Can anyone explain me StandardScaler?, n.d.). To achieve this, we used sklearn pre-defined function StandardScaler. After performing standard scalar our dataset looked something like this.

```
In [18]: from sklearn.preprocessing import StandardScaler

         s_sc = StandardScaler()
         cts = ['age','trestbps', 'chol', 'thalach', 'oldpeak']
         df[cts] = s_sc.fit_transform(df[cts])
         df.head()
```

Out[18]:

	age	trestbps	chol	thalach	oldpeak	target	cp_0	cp_1	cp_2	cp_3	...	exang_0	exang_1	ca_0	ca_1	ca_2	ca_3	ca_4	restecg_0	reste
0	0.952197	0.763956	-0.256334	0.015443	1.087338	1	0	0	0	1	...	1	0	1	0	0	0	0	1	
1	-1.915313	-0.092738	0.072199	1.633471	2.122573	1	0	0	1	0	...	1	0	1	0	0	0	0	0	
2	-1.474158	-0.092738	-0.816773	0.977514	0.310912	1	0	1	0	0	...	1	0	1	0	0	0	0	1	
3	0.180175	-0.663867	-0.198357	1.239897	-0.206705	1	0	1	0	0	...	1	0	1	0	0	0	0	0	
4	0.290464	-0.663867	2.082050	0.583939	-0.379244	1	1	0	0	0	...	0	1	1	0	0	0	0	0	

5 rows × 31 columns

The mathematical equation for standardization is.

$$z = \frac{x - \mu}{\sigma} \tag{1}$$

where

$$\mu = \frac{1}{N}\sum_{i=1}^{N}(x_i) \tag{2}$$

And

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2} \tag{3}$$

3) Splitting the data for Training and Testing:: After all these steps we split the data into 30:70 for testing and training.

```
In [19]: y = df.target.values
         x = df.drop(['target'], axis = 1)

In [20]: x_train, x_test, y_train, y_test = train_test_split(x,y,test_size = 0.3,random_state=44)
```

With this step we conclude our data pre-processing and now we'll move on to training the model and testing it.

III. Experiments and Result Evaluation:

A. Model Training and Testing:

In order to find the best accuracy, we trained this dataset on three models. 1. Decision Tree Classification 2. Random

Forest Classification 3. Support Vector Machine Classification Let's investigate each one of them one by one and see the results.

1) Decision Tree Classification:: So, first off we are importing the decision tree classifier form the sklearn library and then we are training the data set. We are using .score to measure the accuracy, reason for using .score is that we don't have to predict the data, .score does it on its own.

```
from sklearn.tree import DecisionTreeClassifier
dtc = DecisionTreeClassifier()
dtc.fit(x_train, y_train)
acc1 = dtc.score(x_test, y_test)*100
print("Decision Tree Testing Accuracy {:.2f}%".format(acc1))
```

```
Decision Tree Testing Accuracy 76.92%
```

2) Random Forest Classification: Just like before we are importing the random forest classifier and then this time, we are setting number of trees (n_estimator) to 10 and random state to 1. After this we are training the model.

```
# Random Forest Classification
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators = 10, random_state = 1)
rf.fit(x_train, y_train)
acc2 = rf.score(x_test,y_test)*100
print("Random Forest Algorithm Accuracy Score : {:.2f}%".format(acc2))
```

```
Random Forest Algorithm Accuracy Score : 81.32%
```

3) Support Vector Machine Classifier:: Here we can see that we are importing SVC from Sklearn. The only parameter we are passing is the random state which is 1 and then we are training the model. After this we are training the model.

```
from sklearn.svm import SVC
svm = SVC(random_state = 1)
svm.fit(x_train, y_train)
acc3 = svm.score(x_test,y_test)*100
print("SVM Algorithm Accuracy Score: {:.2f}%".format(acc3))
```

```
Test Accuracy of SVM Algorithm: 83.52%
```

B. Value Prediction:

Now that we have trained all the models, let's predict the values.

```
# Predicted values
Y_pred_svm = svm.predict(x_test)
y_pred_dtc = dtc.predict(x_test)
y_pred_rf = rf.predict(x_test)
```

Predicting Values

```
In [28]: # Predicted values
y_pred_svm = svm.predict(x_test)
y_pred_dtc = dtc.predict(x_test)
y_pred_rf = rf.predict(x_test)
```

Random Forest Prediction

```
In [29]: y_pred_rf0 = pd.DataFrame( { "Actual": y_test,  "Predicted": y_pred_rf } )
y_pred_rf0.head()
```

Out[29]:

	Actual	Predicted
0	1	1
1	1	1
2	0	1
3	0	1
4	1	1

Decision Tree Prediction

```
In [30]: y_pred_dtc0 = pd.DataFrame( { "Actual": y_test,  "Predicted": y_pred_dtc } )
y_pred_dtc0.head()
```

Out[30]:

	Actual	Predicted
0	1	0
1	1	0
2	0	0
3	0	0
4	1	1

Support Vector Machine Prediction

```
In [31]: y_pred_svm0 = pd.DataFrame( { "Actual": y_test,  "Predicted": y_pred_svm } )
y_pred_svm0.head()
```

Out[31]:

	Actual	Predicted
0	1	1
1	1	1
2	0	1
3	0	1
4	1	1

C. Accuracy:

Let's look at the Accuracy of all the algorithms and see which one did the best.

Testing Accuracy before Hyper Parameter Tuning

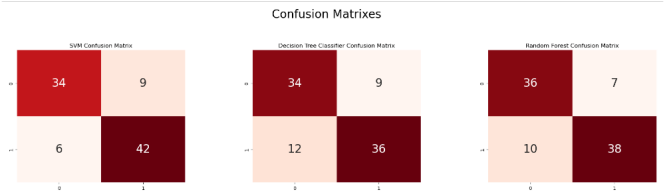
```
results_df_Pre_Opt
```

	Model	Testing Accuracy %
0	DecisionTreeClassifier	76.923077
1	RandomForestClassifier	81.318681
2	Support Vector Machine	83.516484

As we can see off all the algorithms Support Vector Machine did the best with a score of 83.516

D. Confusion Matrix:

A confusion matrix shows how many predictions our models did right and how many they did wrong. Just to help us **upper left is True Positive, Upper Right is false Positive, Lower left is False Negative and Lower Right is True Negative**



We can take following points out of this.

1) SVM got 34 true positive and 42 true negatives, also it got 6 false negative and 9 false positive.
2) Decision Tree got 34 true positive and 36 true negatives, also it got 12 false negative and 9false positive.
3) Random Forest got 34 true positive and 42 true negatives, also it got 6 false negative and 9 false positive.

After all this we can see our model training wasn't perfect. We can still make it better. If you see, we didn't specify any parameters for all the trainings. Now let's try Hyperparameter Tuning.

E. Hyperparameter Tuning:

We tuned all three of our models and the results were rather amazing.

1) Decision Tree Hyperparameter Tuning:: Took help from *https://www.kaggle.com/code/faressayah/predicting-heart-disease-using-machine-learning/notebook*

```
params = {"criterion":("gini", "entropy"),
          "splitter":("best", "random"),
          "max_depth":(list(range(1, 20))),
          "min_samples_split":[2, 3, 4],
          "min_samples_leaf":list(range(1, 20))
          }
dtc_cv = GridSearchCV(dtc, params, n_jobs=-1, verbose=1, cv=3)
dtc_cv.fit(x_train, y_train)
best_params = dtc_cv.best_params_
```

```
Fitting 3 folds for each of 4332 candidates, totalling 12996 fits
```

```
dtc = DecisionTreeClassifier(**best_params)
dtc.fit(x_train, y_train)
acc4 = dtc.score(x_test, y_test)*100
print("Decision Tree Test Accuracy {:.2f}%".format(acc4))
```

```
Decision Tree Test Accuracy 81.32%
```

So here we can see first off, we set some pre-defined parameters and with the help of GridSearchCV we'll find the best set of parameters. Then we'll train the model again and this time around our accuracy is 81.32

F. Random Forest Hyperparameter Tuning:

For this part we took help from *https://www.kaggle.com/code/arjunprasadsarkhel/simple-random-forest-with-hyperparameter-tuning*

```
rf = RandomForestClassifier(**best_params)
rf.fit( x_train, y_train)
acc5 = rf.score(x_test,y_test)*100
print("Random Forest Algorithm Accuracy Score : {:.5f}%".format(acc5))
```

```
Random Forest Algorithm Accuracy Score : 84.61538%
```

1) Support Vector Machine Hyperparameter Tuning: Took help from *https://www.kaggle.com/code/rajeevnair676/svm-hyperparameter-tuning*

```
params = {"C":(0.1, 0.5, 1, 2, 5, 10, 20),
          "gamma":(0.001, 0.01, 0.1, 0.25, 0.5, 0.75, 1),
          "kernel":('linear', 'poly', 'rbf')}

svm_cv = GridSearchCV(svm, params, n_jobs=-1, cv=5, verbose=1,)
svm_cv.fit(x_train, y_train)
best_params = svm_cv.best_params_
```

```
Fitting 5 folds for each of 147 candidates, totalling 735 fits
```

```
## Took help from https://www.kaggle.com/code/rajeevnair676/svm-hyperparameter-tuning
svm = SVC(**best_params)
svm.fit(x_train, y_train)
acc6 = svm.score(x_test,y_test)*100
print("Test Accuracy of SVM Algorithm: {:.5f}%".format(acc6))
```

```
Test Accuracy of SVM Algorithm: 85.71429%
```

G. Prediction after Hyperparameter Tuning

After hyperparameter tuning we can see that our accuracy improved a lot. Now we'll see the predicted values with upgraded models.

Predicting Values

```
In [56]: # Predicted values
y_pred_svm = svm.predict(x_test)
y_pred_dtc = dtc.predict(x_test)
y_pred_rf = rf.predict(x_test)
```

Random Forest Prediction After Tuning

```
In [57]: y_pred_rf0 = pd.DataFrame( { "Actual": y_test,  "Predicted": y_pred_rf } )
y_pred_rf0.head()
```

Out[57]:

	Actual	Predicted
0	1	1
1	1	1
2	0	1
3	0	1
4	1	1

Decision Tree Prediction After Tuning

```
In [58]: y_pred_dtc0 = pd.DataFrame( { "Actual": y_test,  "Predicted": y_pred_dtc } )
y_pred_dtc0.head()
```

Out[58]:

	Actual	Predicted
0	1	1
1	1	1
2	0	0
3	0	1
4	1	1

Support Vector Machine Prediction After Tuning

```
In [59]: y_pred_svm0 = pd.DataFrame( { "Actual": y_test,  "Predicted": y_pred_svm } )
y_pred_svm0.head()
```

Out[59]:

	Actual	Predicted
0	1	1
1	1	1
2	0	1
3	0	1
4	1	1

H. Accuracy: Before and After Hyperparameter Tuning:

Let's compare the accuracy of our models before and after tuning.

Testing Accuracy before Hyper Parameter Tuning

```
: results_df_Pre_Opt
```

:

	Model	Testing Accuracy %
0	DecisionTreeClassifier	76.923077
1	RandomForestClassifier	81.318681
2	Support Vector Machine	83.516484

Testing Accuracy after Hyper Parameter Tuning

```
: results_df_Post_Opt
```

:

	Model	Testing Accuracy %
0	DecisionTreeClassifier	81.318681
1	RandomForestClassifier	84.615385
2	Support Vector Machine	85.714286

As we can see that after tuning Decision Tree went from 76.9 to 81.4, Random Forest went from 81.3 to 84.61 and SVM went from 83.51 to 85.71. The model with the greatest gain is the Decision Tree classifier.

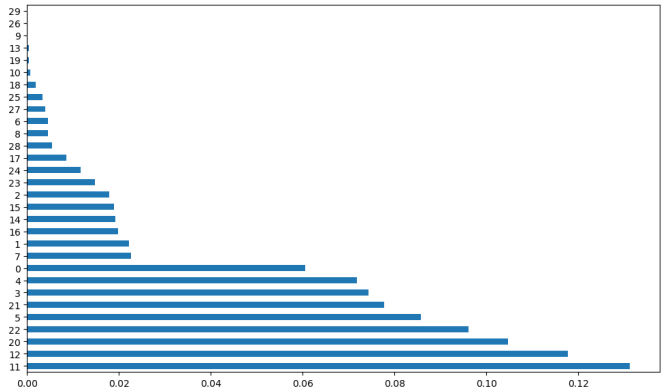
I. Important Features:

With the help of Random Forest and Decision tree Classification we can figure out the most important features according to each model.

```
def feature_imp(df, model):
    fi = pd.DataFrame()
    fi["feature"] = df.columns
    fi["importance"] = model.feature_importances_
    return fi.sort_values(by="importance", ascending=False)
```

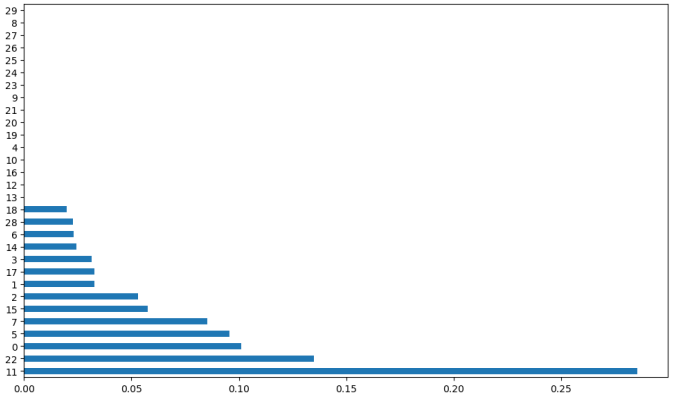
For this function we took help from *https://www.kaggle.com/code/faressayah/predicting-heart-disease-using-machine-learning/notebook*

1) Important features according to Random Forest::



According to Random Forest the most important feature is Feature 11 which is Thal.

2) Important features according to Decision Tree:::



According to Random Forest the most important feature is Feature 11 which is Thal.

IV. Conclusion and Findings:

After all the testing we can say that the Support Vector Machine is the most efficient and accurate model. Moreover feature 11 and 12 which are Thalassemia are the most important features. We tried and tested different things in the whole experiment. First off, we noted that lower number of n_estimators give higher accuracy. As for SVM random_state and accuracy are inversely proportional. One thing that I noted after trying the same code with normalisation instead of standardisation is that the accuracy was all over the place with normalisation. Accuracy with normalisation.

Testing Accuracy before Hyper Parameter Tuning

```
]: results_df_Pre_Opt
```

]:

	Model	Testing Accuracy %
0	DecisionTreeClassifier	74.725275
1	RandomForestClassifier	81.318681
2	Support Vector Machine	86.813187

Testing Accuracy after Hyper Parameter Tuning

```
]: results_df_Post_Opt
```

]:

	Model	Testing Accuracy %
0	DecisionTreeClassifier	76.923077
1	RandomForestClassifier	81.318681
2	Support Vector Machine	83.516484

Accuracy with standardisation.

Testing Accuracy before Hyper Parameter Tuning

```
: results_df_Pre_Opt
```

:

	Model	Testing Accuracy %
0	DecisionTreeClassifier	76.923077
1	RandomForestClassifier	81.318681
2	Support Vector Machine	83.516484

Testing Accuracy after Hyper Parameter Tuning

```
: results_df_Post_Opt
```

:

	Model	Testing Accuracy %
0	DecisionTreeClassifier	81.318681
1	RandomForestClassifier	84.615385
2	Support Vector Machine	85.714286

References

[1] Can anyone explain me StandardScaler? (n.d.). Retrieved from Stack Overflow: https://stackoverflow.com/questions/40758562/can-anyone-explain-me-standardscaler

[2] https://www.kaggle.com/code/faressayah/predicting-heart-disease-using-machine-learning/notebook

[3] https://www.kaggle.com/code/cdabakoglu/heart-disease-classifications-machine-learning/notebook

[4] https://www.kaggle.com/code/arjunprasadsarkhel/simple-random-forest-with-hyperparameter-tuning

[5] https://www.kaggle.com/code/rajeevnair676/svm-hyperparameter-tuning/notebook