Heart disease prediction using Machine learning techniques

First Author#, Second Author\*, Third Author#

#First-Third Department, First-Third University  
Address

1first.author@first-third.edu

3third.author@first-third.edu

\*Second Company  
 Address Including Country Name

2second.author@second.com

Abstract— Heart diseases or cardiac arrest has emerged as major killers in current society and this is become more problem in poor Nations in last few years as per surveys, the number of heart-related diseases is more than 75% higher in current days in all those poor nations so in this paper we are going to study about the reasons of the cardiac arrest and the causes and some other features which are causing cardiac arrests by gathering data from different sources and doing analysis on the collected data and extract multiple understanding and insights that causes cardiac arrests also it is very important to detect the heart disease early so as part of this process we are going to analyse the different cases where patients are getting this heart diseases.

Keywords— Machine learning, Cardiac arrest, Heart disease, Logistic regression, Random Forest classifier, Decision tree classifier.

1. Introduction

[1] Heart disease refers to a group of illnesses that damage your heart. Blood vessel disorders such as coronary artery disease, heart rhythm issues (arrhythmias), and birth defects (congenital heart defects) are examples of heart diseases.

[1] The terms "heart disease" and "cardiovascular disease" are frequently used interchangeably. Cardiovascular disease refers to disorders characterised by restricted or obstructed blood arteries, which can result in a heart attack, chest discomfort (angina), or stroke. Other cardiac problems, such as those affecting your heart's muscle, valves, or rhythm, are also classified as heart disease.

[1] Heart disease is one of the leading causes of illness and mortality in the world's population. One of the most significant topics in clinical data analysis is the prediction of cardiovascular disease. The healthcare business generates a massive quantity of data. Data mining converts a significant amount of raw healthcare data into information that can be used to make educated choices and forecasts.

[2] Cardiovascular disease is the main cause of mortality among the top ten causes of death globally, according to the World Health Organization's 2019 report. Between 2000 (11%) and 2019 (16%), the percentage of patients with heart disease increased dramatically. Consider that two million people died as a result of it in 2002, and that figure is expected to rise to 8.9 million in 2019. According to data, the global risk of CVDs has grown in recent decades and has become more common in developing and underdeveloped countries. Much study is being conducted in an effort to reveal important components in heart disease and clearly predict overall risk in order to appreciate the mechanism underlying it and minimise the total risk. In this fast-paced setting, people build themselves luxury while ignoring healthy behaviours, which lead to high blood pressure and glucose levels in adolescents and present a challenging problem: heart illness. Heart disease can be identified by signs including high blood pressure, chest discomfort, tension headaches, heart attacks, and lingering symptoms.

[2] There is several different kinds of heart diseases, each with its own set of symptoms. very high blood pressure, upper body aches, shock, and cardiac fluid illness are all symptoms. A slow pulse, uneasiness, and chest pain are signs of heart disease brought on by irregular heartbeats. high blood pressure, upper body pain, excessive stress, cardiac arrest, and other symptoms. Because it kills without causing symptoms, heart disease is sometimes referred to as the "quiet predator.". To maintain a healthy lifestyle, it is essential to identify heart disease early in high-risk patients. To expect potential heart infection, this study uses artificial intelligence algorithms to classify patients as having heart disease.

[2] cardiovascular diseases (CVDs), which are heart-related disorders, have recently emerged as the most dangerous illness in both India and the rest of the world and are the leading cause of death globally. It is therefore necessary to diagnose these illnesses early enough for successful therapy, and this requires a reliable, accurate, and practical procedure. ML methods are applicable to multiple medical field related collected data can be used to automate and do very complex tasks with ease. Many Research scientists are currently utilizing the different ML techniques to guide and identifying heart-related illnesses.

1. Problem and Data set(s)

The problem of the data set to identify heart disease and what are the factors that affect heart disease. So, in our original data set we are having 76 attributes and here we are only considering 14 attributes for our analysis. The independent variables are age, sex, chest pain type, resting blood pressure, cholesterol, fasting blood sugar, maximum heart rate, exercise induced angina, oldpeak, resting electrocardiographic results, the slope of the peak exercise ST segment and the dependent variable is the target which has information whether the person is having heart disease or not. Our problem statement is the binary classification problem statement.

1. Methods

Here for our analysis, we are considering three machine learning algorithms those are logistic regression k nearest neighbour and decision tree the in-detail explanation of each algorithm is given below:

## Logistic regression:

Logistic regression is one of the most used machine learning algorithms which takes label data as input and classify the output label basically logistic regression is used to solve classification problems and it will take multiple independent variables and one dependent variable as the output for training and do classification. basically, regression predicts output of categorical dependent variable so hear the output will be in the form of either yes or no 0 or 1 or binary output. So, instead of giving except 0 or 1 it will return in numerical value and with help of Sigmoid function will convert this numerical value between zero or one. The logistic regression is almost like linear regression which is used for regression techniques except the last layer which consist of this probability layer which converts regular value into binary value. Logistic regression instead of fitting regression line we will fit a Sigma curve, this sigmoid curve will help us to generate probabilities values from regression values also the curve from logistic function explains the likelihood of our problem statement like whether image is cat or not, patient is pregnant or not. The assumption of logistic regression includes the depend on and variable must be categorical and independent variables do not have multi collinearity. The only difference between linear regression and logistic regression is logistic regression uses the concept of predictive modelling like linear regression but it is used to classify samples so it will fall under classification algorithm.

## KNN:

KNN also known as k nearest neighbour is a machine learning algorithm that is very simple and falls under supervised learning this can be used to do regression tests or classification tasks and also majority this week can be used for filling the missing values or missing values immigration the working principle behind this care nearest algorithm is that the observations close to a given data point or the most similar observations given in the data set and therefore we can classify your predict unseen points of the closest existing points so choose the number of nearby points to look after we can select those close values with help of k value. KNN is a non-parametric and lazy learner algorithm because it does not make any assumptions on underlined data so we call it nonparametric it does not learn anything from given data adjust observes and give us insights and classify the given points so we will call it lazy learning also can an algorithm stores the data instead of learning, and whenever a new sample comes it will just compare and give the results so we can call it as an algorithm operating of KNN is very much simple unlike other algorithms as part of this first we need to select the number of k nearest neighbours and then we have to calculate Euclidian distance for the selected k number of neighbours then we have to take care nearest neighbours as per calculated Euclid and distance and count the data points in each category class signed the new data points to its respective category based on the distance it is having.

## Decision tree:

Decision tree is also a supervisor learning technique which will performs both regression and classification task for giving data it is a tree structured classifier where all internal nodes or columns of data set and branches explains the decision rules that is taken by the decision tree of outcome basically in a decision tree we will be having to notes those are the output of the decision note and this leaf notes will not be having any further branches but decision notes used to take many decisions which will be having multiple branches also decision tree is one of the most beautiful algorithm which is having graphical representation of all its working and we can visualise the decision tree once the model is win ready reasons for calling this decision and this expands for the branches from root note to construct structure like this so for building decision tree we will be using cart algorithm which means classification and regression algorithm decision tree can contain both numerical and category data the reason for using decision trees is the log behind decision tree can be easily understood and visualizes because it's a tree like structure and decision tree usually works similar to human thinking ability while making a decision so it is very much easier to understand the working principle of decision tree the operation of decision tree is very much simple so first we have to select the given complete data set then we have to find best attribute in the given data set using attribute selection measure then we have to divide the road not into subset that contains possible values for best attributes and after splitting we have to generate decision tree no which contains best attribute and we have to do this process recursively to make new decisions and aunty should be leaf node. For selecting root node, we can use either information gain or Gini index and, we can apply pruning to avoid the risk of over fitting.

## Random Forest classifier:

Random forest is a popular machine learning algorithm that comes under the branch of supervised machine learning so this algorithm can be utilized for both classification and regression problems in machine learning also this is a part of ensemble learning as we will create multiple decision trees under each algorithm basically ran over is the process of combining multiple classifiers to solve a complex problem and improve the performance of the model. Random forest is defined as a classifier that contains the different number of decision tree is which belongs to various subsets of the given data and takes average of the predative accuracy of the data set so the more number of decision tree is we built the more accuracy we will get there are few assumptions related to this random forest they are predictions from each decision tree will be having least core relation and there should be some actual values in the future variable of the data set so that random forest can give accurate results rather than then random result the reasons for choosing random for s for this problem is it takes less training term and gives more accurate result compare to other algorithms also the output produced by random forest is having high accuracy even for larger data set also and it maintains stability in the output the working of random forest algorithm is as follows first will select random K point for training set and we will build decision trees associated with the given data of subsets then we will choose number of decision trees that needs to be built and we will repeat above steps for multiple times at last will export the model and for new data points we can find the predictions of new data points to the category that it belongs to base run the majority of vote that it got the few advantages of random forest algorithm is it is performing both classification and regression task and it is capable of handling high dimensional problems with the large data sets it enhance is the currency of model and prevent the over fitting issue all though random forest can be used for both regression and classification tasks it is not recommended to use for regression tasks in majority

1. Experimental setup

As part of exploratory data analysis, we are going to use seaborn and matplotlib. Also our data cities consist of 300 records and 14 columns and our target variable is consists of zeros and once and the count of each are seems to be balanced this can be observed with help of bar plot that shown in below diagram and out of 3003 records there is zero missing values found and there is no correlated variables also the least coefficient factor where having is minus 0.4 where the highest positive correlation factor where having is 0.43 and some of the visualisations we did with respect to our target variable and gender so few highlights of this kind of analysis are follows mail please having more chances of getting hard disease than female and the persons who is having chest pain can get a hard disease than people without chest pain and the persons who is having age between 50 to 70 is having more chances of getting heart disease and the persons who is having cholesterol ranging between 200 to 300 is having a more chance of getting a disease and there is no outliers present in our data so as part of data pre-processing we have to change few column names based on the category city having as part of that first we are going to change just pain column and we are going to split this into four columns as it is having four categories so after doing this pre-processing we used one heart encoding for encoding all categorical variables so now our data frame is in complete numerical format and we can use it for building models to classify whether the person is having heart disease or not.

1. Results

So from our pre-processing data, our data is completely clean and we can utilise this available data for preparing the models so before start training the models we have to split over data for our model evaluation so we are converting 80% of data into training and 20% of data into testing and here to find the patterns in training set where going to try for different machine learning models those are logistic regression k Nearest neighbour classifier decision tree classifier, random forest classifier and for the first iteration if we see we got 88% model score for logistic regression 68% model score for k nearest neighbour 86% model score for random forest classifier and 81% score for this classifier so the visualisation of this model comparison can be observed from below figure.

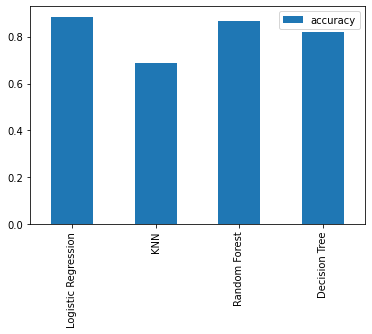


Fig: Accuracy scores of different machine learning models.

And now it's time to do hyper parameter during of k nearest neighbour classifier as it scored very much less and after hyperparameter tuning the maximum score that we can obtain from test data is 75% and now it time to do hyperparameter tuning for logistic regression and random forest classifier algorithms as they are good performing algorithms. After applying randomized search cross-validation for logistic regression we got a 90% model score whereas earlier the score is nearer to 88% now we applied the same for random for s classifier and we got 88.5% with randomized search cross-validation, whereas earlier it is 86%.

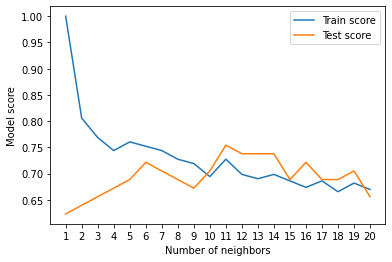


Fig: Different Model scores for different K values for K nearest neighbour model.

Now apart from accuracy we tried plotting area under the curve for the logistic regression model as it is giving the best results and good models scores compared to other models, this can be observed from Below figure.

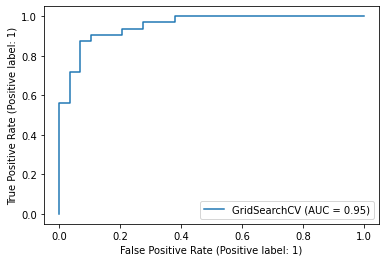


Fig: AUC plot for Logistic regression with Grid search CV

Also, we observed it is having a smaller number of false positives and false negatives when compared to other algorithms and, we can observe it is having a very good amount of precision, recall F1 score values and finally we applied cross-validation also and the model is still performing very well so we selected logistic regression classified as our final model for further process. The precision, recall and F1 score can be observed from below figure.

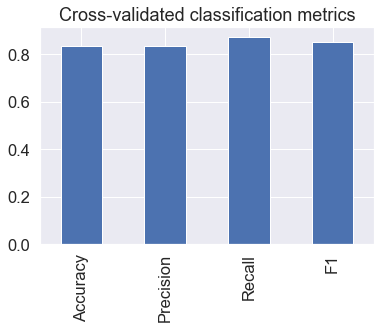


Fig: Accuracy, Precision and recall plot for Logistic regression.

1. Conclusions

The At last we concluded that logistic regression model is performing better than any other model we tried but as an extension might be adding the matrix on top of each model may help us to judge the performance better and the final thing is to check of the list of our model evaluation techniques is feature importance according to the model there is a positive correlation of 0.47 not as strong as 6 and target but still more than zero this positive correlation means our model is speaking of the pattern as that slope is increase so it affects the target value also we can do some more things like if some of the correlation and feature important are confusing be may have to drop that variables for better model performance and in our data the number of records and features are very less so it is not sufficient to learn the model so it is better to get some more data for good performing model and here are some of the experimentation techniques that we can try is from trying different models to tuning different models to figure it out which hyperparameters purpose as we applied different hyperparameters to different models it is always best to apply suitable hyperparameters for suitable models and we know the current model where using tuned version of logistic regression along with our specific data set doesn't hit target reset ourselves but adding more to about data which will help model to predict and classify better so next steps we can collect more data and train again are we have to choose any & model like XGboost, are still trying for more fine-tuned model which can perform even more better.

1. References

[1]

“Heart Disease Prediction using Machine Learning,” *Analytics Vidhya*, Feb. 11, 2022. https://www.analyticsvidhya.com/blog/2022/02/heart-disease-prediction-using-machine-learning/

[2]

I. Journal, “Heart Disease Prediction Using Machine Learning Algorithms,” *INTERANTIONAL JOURNAL OF SCIENTIFIC RESEARCH IN ENGINEERING AND MANAGEMENT*, vol. 06, no. 06, Jun. 2022, doi: 10.55041/ijsrem14688.

[3]

R. Baxani and M. Edinburgh, “Heart Disease Prediction Using Machine Learning Algorithms Logistic Regression, Support Vector Machine and Random Forest Classification Techniques,” *SSRN Electronic Journal*, 2022, doi: 10.2139/ssrn.4151423.

[4]

R. Goel, “Heart Disease Prediction Using Various Algorithms of Machine Learning,” *SSRN Electronic Journal*, 2021, doi: 10.2139/ssrn.3884968

Appendix:

# Import all the tools we need

# Regular EDA (exploratory data analysis) and plotting libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# we want our plots to appear inside the notebook

get\_ipython().run\_line\_magic('matplotlib', 'inline')

# Models from Scikit-Learn

from sklearn.linear\_model import LogisticRegression

from sklearn.neighbors import KNeighborsClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.tree import DecisionTreeClassifier

# Model Evaluations

from sklearn.model\_selection import train\_test\_split, cross\_val\_score

from sklearn.model\_selection import RandomizedSearchCV, GridSearchCV

from sklearn.metrics import confusion\_matrix, classification\_report

from sklearn.metrics import precision\_score, recall\_score, f1\_score

from sklearn.metrics import plot\_roc\_curve

# ## 2.Load Data

df = pd.read\_csv("heart-disease.csv")

df.shape # (rows, columns)

# ## 3.Data Exploration (exploratory data analysis or EDA)

df.head()

df.tail()

# Let's find out how many of each class there

df["target"].value\_counts()

# Since these two values are close to even, our target column can be considered balanced. An unbalanced target column, meaning some classes have far more samples, can be harder to model than a balanced set. Ideally, all of your target classes have the same number of samples.

#

#

# We can plot the target column value counts by calling the plot() function and telling it what kind of plot we'd like, in this case, bar is good.

# In[6]:

df["target"].value\_counts().plot(kind="bar", color=["salmon", "lightblue"]);

# df.info() shows a quick insight to the number of missing values you have and what type of data your working with.

#

# In our case, there are no missing values and all of our columns are numerical in nature.

# In[7]:

df.info()

# In[8]:

# Are there any missing values?

df.isna().sum()

# In[9]:

# visualizing null values if it exists

plt.figure(figsize=(22,10))

plt.xticks(size=20,color='grey', rotation = 45)

plt.tick\_params(size=12,color='white')

plt.title('Finding Null Values Using Heatmap\n', color='white',size=30)

sns.heatmap(df.isnull(), yticklabels=False, cbar=True, cmap='PuBu\_r')

# In[10]:

df.describe()

# In[11]:

# Let's make our correlation matrix a little prettier

corr\_matrix = df.corr()

fig, ax = plt.subplots(figsize=(15, 10))

ax = sns.heatmap(corr\_matrix,

annot=True,

linewidths=0.5,

fmt=".2f",

cmap="YlGnBu");

bottom, top = ax.get\_ylim()

ax.set\_ylim(bottom + 0.5, top - 0.5)

# ## 4.Data Visuvalization

# In[12]:

# Creating a copy of original dataset

df2 = df.copy()

# #### 4.1. Count plot

# In[13]:

def chng(sex):

if sex == 0:

return 'female'

else:

return 'male'

# Applying the function for "Sex" column

df2['sex'] = df2['sex'].apply(chng)

# In[14]:

def chng(sex):

if sex == 0:

return 'female'

else:

return 'male'

# Applying the function for "Sex" column

df2['sex'] = df2['sex'].apply(chng)

def chng2(prob):

if prob == 0:

return 'Heart Disease'

else:

return 'No Heart Disease'

# Applying the function for Target function

df2['target'] = df2['target'].apply(chng2)

# In[15]:

plt.figure(figsize=(16,7))

sns.countplot(data= df2, x='sex', hue='target')

plt.title('Gender v/s target')

# In[16]:

plt.figure(figsize=(16,7))

sns.countplot(data= df2, x='cp', hue='target')

plt.title('Chest Pain Type v/s target')

# In[17]:

plt.figure(figsize=(16,7))

sns.countplot(data= df2, x='sex',hue='thal')

plt.title('Gender v/s Thalassemia')

# In[18]:

plt.figure(figsize=(16,7))

sns.countplot(data= df2, x='slope',hue='target')

plt.title('Slope v/s Target')

# In[19]:

plt.figure(figsize=(16,7))

sns.countplot(data= df2, x='exang',hue='thal')

plt.title('exang v/s Thalassemia')

# #### 4.2. Distplot

#

# In[20]:

# Age of Patients having heart disease

age\_tar\_0 = df[df['target']==0]['age']

plt.figure(figsize=(16,7))

sns.distplot(age\_tar\_0, kde=True, bins=50)

plt.title('Normal Distribution of Age of Heart Diseased Patients')

# In[21]:

# Chol level for patients with heart disease

chol\_tar\_0 = df[df['target']==0]['chol']

# Chol level of the patients

plt.figure(figsize=(16,7))

sns.distplot(chol\_tar\_0, kde=True, bins=50)

plt.title('Normal Distribution for Chol of Heart Diseased Patients')

# In[22]:

# Thalach level for patients with heart disease

Thalach\_tar\_0 = df[df['target']==0]['chol']

plt.figure(figsize=(16,7))

sns.distplot(Thalach\_tar\_0,kde=True, bins=50)

plt.title('thalach of Heart Diseased Patients')

# #### 4.3. Jointplot

#

# In[23]:

# Selecting all patients having heart disease

df3 = df[df['target'] == 0 ][['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',

'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target']]

df3.head()

# In[24]:

pal = sns.light\_palette("blue", as\_cmap=True)

plt.figure(figsize=(20,10))

print('Age vs trestbps(Heart Diseased Patinets)')

sns.jointplot(data=df3, x='age', y='trestbps', kind='hex', cmap='Reds')

plt.show()

# In[25]:

sns.jointplot(data=df3, x='chol', y='age', kind='kde', cmap='PuBu')

# In[26]:

sns.jointplot(data=df3, x='chol', y='trestbps', kind='resid')

# #### 4.4. Boxplot

#

# In[27]:

sns.boxplot(data=df2,x='target',y='age')

# In[28]:

plt.figure(figsize=(10,7))

sns.boxplot(data=df2,x='cp',y='thalach',hue='target')

# In[29]:

plt.figure(figsize=(10,7))

sns.boxplot(data=df2,x='fbs',y='trestbps',hue='target')

# In[30]:

plt.figure(figsize=(10,7))

sns.boxplot(data=df2,x='slope',y='thalach',hue='target')

# ## 5. Data Pre-processing

#

# ### 5.1 Change Name of the column

#

# In[31]:

df.columns = ['age', 'sex', 'chest\_pain\_type', 'resting\_blood\_pressure', 'cholesterol', 'fasting\_blood\_sugar',

'rest\_ecg\_type', 'max\_heart\_rate\_achieved', 'exercise\_induced\_angina', 'st\_depression', 'st\_slope\_type',

'num\_major\_vessels', 'thalassemia\_type', 'target']

df.columns

# For the given dataset, we have 4 Categorical Variables and rest are Numerical Values

#

# cp - chest\_pain\_type

# restecg - rest\_ecg\_type

# slope - st\_slope\_type

# thal - thalassemia\_type

# In[32]:

# Unique values in categorical values

list = []

cols = ['chest\_pain\_type', 'rest\_ecg\_type', 'st\_slope\_type', 'thalassemia\_type']

for i in cols:

uniq\_val\_cat = len(df[i].unique())

print("Column:",i)

print(df[i].unique())

list.append(uniq\_val\_cat)

# Printing unique values for the selected columns

df\_uni\_val\_cat = pd.DataFrame({'columns': cols, 'unique\_values': list})

df\_uni\_val\_cat

# #### Converting Numerical Values to Categorical Values

#

# In[33]:

df\_prep = df.copy()

df\_prep.head(5)

# In[34]:

# cp - chest\_pain\_type

df\_prep.loc[df\_prep['chest\_pain\_type'] == 0, 'chest\_pain\_type'] = 'asymptomatic'

df\_prep.loc[df\_prep['chest\_pain\_type'] == 1, 'chest\_pain\_type'] = 'atypical angina'

df\_prep.loc[df\_prep['chest\_pain\_type'] == 2, 'chest\_pain\_type'] = 'non-anginal pain'

df\_prep.loc[df\_prep['chest\_pain\_type'] == 3, 'chest\_pain\_type'] = 'typical angina'

# restecg - rest\_ecg\_type

df\_prep.loc[df\_prep['rest\_ecg\_type'] == 0, 'rest\_ecg\_type'] = 'left ventricular hypertrophy'

df\_prep.loc[df\_prep['rest\_ecg\_type'] == 1, 'rest\_ecg\_type'] = 'normal'

df\_prep.loc[df\_prep['rest\_ecg\_type'] == 2, 'rest\_ecg\_type'] = 'ST-T wave abnormality'

# slope - st\_slope\_type

df\_prep.loc[df\_prep['st\_slope\_type'] == 0, 'st\_slope\_type'] = 'downsloping'

df\_prep.loc[df\_prep['st\_slope\_type'] == 1, 'st\_slope\_type'] = 'flat'

df\_prep.loc[df\_prep['st\_slope\_type'] == 2, 'st\_slope\_type'] = 'upsloping'

# thal - thalassemia\_type

df\_prep.loc[df\_prep['thalassemia\_type'] == 0, 'thalassemia\_type'] = 'nothing'

df\_prep.loc[df\_prep['thalassemia\_type'] == 1, 'thalassemia\_type'] = 'fixed defect'

df\_prep.loc[df\_prep['thalassemia\_type'] == 2, 'thalassemia\_type'] = 'normal'

df\_prep.loc[df\_prep['thalassemia\_type'] == 3, 'thalassemia\_type'] = 'reversable defect'

# ### 5.2 One Hot Encoding

#

# In[35]:

data = pd.get\_dummies(df\_prep, drop\_first=False)

data.head(10)

# In[36]:

df\_temp = data['thalassemia\_type\_fixed defect']

df\_temp

# Since one hot encoding dropped "thalassemia\_type\_fixed defect" column which was a useful column compared to 'thalassemia\_type\_nothing' which is a null column, we dropped 'thalassemia\_type\_nothing' and concatinated 'thalassemia\_type\_fixed defect'

# In[37]:

frames = [data, df\_temp]

result = pd.concat(frames,axis=1)

result.head()

# In[38]:

result.drop('thalassemia\_type\_nothing', axis=1, inplace=True)

resultc = result.copy()

resultc.head()

# ## 6. Modelling

#

# In[39]:

df.head()

# We're trying to predict our target variable using all of the other variables.

#

# To do this, we'll split the target variable from the rest.

# In[40]:

# Split data into X and y

X = resultc.drop("target", axis=1)

y = resultc["target"]

# In[41]:

X.head()

# ### 6.1.Training and test split

#

# In[42]:

# Split data into train and test sets

np.random.seed(42)

# Split into train & test set

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,

y,

test\_size=0.2)

# In[43]:

X\_train

# ### 6.2.Model choices

#

# Now we've got our data split into training and test sets, it's time to build a machine learning model.

#

# We'll train it (find the patterns) on the training set.

#

# And we'll test it (use the patterns) on the test set.

#

# We're going to try 4 different machine learning models:

#

# \* Logistic Regression

# \* K-Nearest Neighbours Classifier

# \* Random Forest Classifier

# \* Decision tree Classifier

# In[44]:

# Put models in a dictionary

models = {"Logistic Regression": LogisticRegression(),

"KNN": KNeighborsClassifier(),

"Random Forest": RandomForestClassifier(),

"Decision Tree": DecisionTreeClassifier()}

# Create a function to fit and score models

def fit\_and\_score(models, X\_train, X\_test, y\_train, y\_test):

"""

Fits and evaluates given machine learning models.

models : a dict of differetn Scikit-Learn machine learning models

X\_train : training data (no labels)

X\_test : testing data (no labels)

y\_train : training labels

y\_test : test labels

"""

# Set random seed

np.random.seed(42)

# Make a dictionary to keep model scores

model\_scores = {}

# Loop through models

for name, model in models.items():

# Fit the model to the data

model.fit(X\_train, y\_train)

# Evaluate the model and append its score to model\_scores

model\_scores[name] = model.score(X\_test, y\_test)

return model\_scores

# In[45]:

model\_scores = fit\_and\_score(models=models,

X\_train=X\_train,

X\_test=X\_test,

y\_train=y\_train,

y\_test=y\_test)

model\_scores

# ### 6.3.Model Comparison

#

# Since we've saved our models scores to a dictionary, we can plot them by first converting them to a DataFrame.

# In[46]:

model\_compare = pd.DataFrame(model\_scores, index=["accuracy"])

model\_compare.T.plot.bar();

# ### 6.4. Hyperparameter tuning (by hand)

# In[47]:

# Let's tune KNN

train\_scores = []

test\_scores = []

# Create a list of differnt values for n\_neighbors

neighbors = range(1, 21)

# Setup KNN instance

knn = KNeighborsClassifier()

# Loop through different n\_neighbors

for i in neighbors:

knn.set\_params(n\_neighbors=i)

# Fit the algorithm

knn.fit(X\_train, y\_train)

# Update the training scores list

train\_scores.append(knn.score(X\_train, y\_train))

# Update the test scores list

test\_scores.append(knn.score(X\_test, y\_test))

# In[48]:

plt.plot(neighbors, train\_scores, label="Train score")

plt.plot(neighbors, test\_scores, label="Test score")

plt.xticks(np.arange(1, 21, 1))

plt.xlabel("Number of neighbors")

plt.ylabel("Model score")

plt.legend()

print(f"Maximum KNN score on the test data: {max(test\_scores)\*100:.2f}%")

# ### 6.5.Hyperparameter tuning with RandomizedSearchCV

#

# We're going to tune:

#

# \* LogisticRegression()

# \* RandomForestClassifier()

#

#

# ... using RandomizedSearchCV

# In[49]:

# Create a hyperparameter grid for LogisticRegression

log\_reg\_grid = {"C": np.logspace(-4, 4, 20),

"solver": ["liblinear"]}

# Create a hyperparameter grid for RandomForestClassifier

rf\_grid = {"n\_estimators": np.arange(10, 1000, 50),

"max\_depth": [None, 3, 5, 10],

"min\_samples\_split": np.arange(2, 20, 2),

"min\_samples\_leaf": np.arange(1, 20, 2)}

# Now we've got hyperparameter grids setup for each of our models, let's tune them using RandomizedSearchCV...

# In[50]:

# Tune LogisticRegression

np.random.seed(42)

# Setup random hyperparameter search for LogisticRegression

rs\_log\_reg = RandomizedSearchCV(LogisticRegression(),

param\_distributions=log\_reg\_grid,

cv=5,

n\_iter=20,

verbose=True)

# Fit random hyperparameter search model for LogisticRegression

rs\_log\_reg.fit(X\_train, y\_train)

# In[51]:

rs\_log\_reg.best\_params\_

# In[52]:

rs\_log\_reg.score(X\_test, y\_test)

# Now we've tuned LogisticRegression(), let's do the same for RandomForestClassifier()...

# In[53]:

# Setup random seed

np.random.seed(42)

# Setup random hyperparameter search for RandomForestClassifier

rs\_rf = RandomizedSearchCV(RandomForestClassifier(),

param\_distributions=rf\_grid,

cv=5,

n\_iter=20,

verbose=True)

# Fit random hyperparameter search model for RandomForestClassifier()

rs\_rf.fit(X\_train, y\_train)

# In[54]:

# Find the best hyperparameters

rs\_rf.best\_params\_

# In[55]:

# Evaluate the randomized search RandomForestClassifier model

rs\_rf.score(X\_test, y\_test)

# ### 6.6.Hyperparamter Tuning with GridSearchCV

#

#

# Since our LogisticRegression model provides the best scores so far, we'll try and improve them again using GridSearchCV...

# In[56]:

# Different hyperparameters for our LogisticRegression model

log\_reg\_grid = {"C": np.logspace(-4, 4, 30),

"solver": ["liblinear"]}

# Setup grid hyperparameter search for LogisticRegression

gs\_log\_reg = GridSearchCV(LogisticRegression(),

param\_grid=log\_reg\_grid,

cv=5,

verbose=True)

# Fit grid hyperparameter search model

gs\_log\_reg.fit(X\_train, y\_train);

# In[57]:

# Check the best hyperparmaters

gs\_log\_reg.best\_params\_

# In[58]:

# Evaluate the grid search LogisticRegression model

gs\_log\_reg.score(X\_test, y\_test)

# ### 6.7.Evaluting our tuned machine learning classifier, beyond accuracy

# In[59]:

# Make predictions with tuned model

y\_preds = gs\_log\_reg.predict(X\_test)

# In[60]:

# Plot ROC curve and calculate and calculate AUC metric

plot\_roc\_curve(gs\_log\_reg, X\_test, y\_test)

# In[61]:

# Confusion matrix

print(confusion\_matrix(y\_test, y\_preds))

# In[62]:

sns.set(font\_scale=1.5)

def plot\_conf\_mat(y\_test, y\_preds):

"""

Plots a nice looking confusion matrix using Seaborn's heatmap()

"""

fig, ax = plt.subplots(figsize=(3, 3))

ax = sns.heatmap(confusion\_matrix(y\_test, y\_preds),

annot=True,

cbar=False)

plt.xlabel("True label")

plt.ylabel("Predicted label")

bottom, top = ax.get\_ylim()

ax.set\_ylim(bottom, top)

plot\_conf\_mat(y\_test, y\_preds)

# Now we've got a ROC curve, an AUC metric and a confusion matrix, let's get a classification report as well as cross-validated precision, recall and f1-score.

# In[63]:

print(classification\_report(y\_test, y\_preds))

# ### 6.8 Calculate evaluation metrics using cross-validation

#

#

# We're going to calculate accuracy, precision, recall and f1-score of our model using cross-validation and to do so we'll be using cross\_val\_score().

# In[64]:

# Check best hyperparameters

gs\_log\_reg.best\_params\_

# In[65]:

# Create a new classifier with best parameters

clf = LogisticRegression(C=0.20433597178569418,

solver="liblinear")

# In[66]:

# Cross-validated accuracy

cv\_acc = cross\_val\_score(clf,

X,

y,

cv=5,

scoring="accuracy")

cv\_acc

# In[67]:

cv\_acc = np.mean(cv\_acc)

cv\_acc

# In[68]:

# Cross-validated precision

cv\_precision = cross\_val\_score(clf,

X,

y,

cv=5,

scoring="precision")

cv\_precision=np.mean(cv\_precision)

cv\_precision

# In[69]:

# Cross-validated recall

cv\_recall = cross\_val\_score(clf,

X,

y,

cv=5,

scoring="recall")

cv\_recall = np.mean(cv\_recall)

cv\_recall

# In[70]:

# Cross-validated f1-score

cv\_f1 = cross\_val\_score(clf,

X,

y,

cv=5,

scoring="f1")

cv\_f1 = np.mean(cv\_f1)

cv\_f1

# In[71]:

# Visualize cross-validated metrics

cv\_metrics = pd.DataFrame({"Accuracy": cv\_acc,

"Precision": cv\_precision,

"Recall": cv\_recall,

"F1": cv\_f1},

index=[0])

cv\_metrics.T.plot.bar(title="Cross-validated classification metrics",

legend=False);