# Predicting heart disease using machine learning

This notebook looks into using various Python-based machine learning and data science libraries in an attempt to build a machine learning model capable of predicting whether or not someone has heart disease based on their medical attributes.

We're going to take the following approach:

1. Problem definition

2. Data

3. Evaluation

4. Features

5. Modelling

6. Experimentation

## 1. Problem Definition

In a statement,

> Given clinical parameters about a patient, can we predict whether or not they have heart disease?

## 2. Data

The original data came from the Cleavland data from the UCI Machine Learning Repository. https://archive.ics.uci.edu/ml/datasets/heart+Disease

There is also a version of it available on Kaggle. https://www.kaggle.com/ronitf/heart-disease-uci

## 3. Evaluation

> If we can reach 95% accuracy at predicting whether or not a patient has heart disease during the proof of concept, we'll pursue the project.

## 4. Features

This is where you'll get different information about each of the features in your data. You can do this via doing your own research (such as looking at the links above) or by talking to a subject matter expert (someone who knows about the dataset).

\*\*Create data dictionary\*\*

1. age - age in years

2. sex - (1 = male; 0 = female)

3. cp - chest pain type

\* 0: Typical angina: chest pain related decrease blood supply to the heart

\* 1: Atypical angina: chest pain not related to heart

\* 2: Non-anginal pain: typically esophageal spasms (non heart related)

\* 3: Asymptomatic: chest pain not showing signs of disease

4. trestbps - resting blood pressure (in mm Hg on admission to the hospital) anything above 130-140 is typically cause for concern

5. chol - serum cholestoral in mg/dl

\* serum = LDL + HDL + .2 \* triglycerides

\* above 200 is cause for concern

6. fbs - (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)

\* '>126' mg/dL signals diabetes

7. restecg - resting electrocardiographic results

\* 0: Nothing to note

\* 1: ST-T Wave abnormality

\* can range from mild symptoms to severe problems

\* signals non-normal heart beat

\* 2: Possible or definite left ventricular hypertrophy

\* Enlarged heart's main pumping chamber

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 looks at stress of heart during excercise unhealthy heart will stress more

11. slope - the slope of the peak exercise ST segment

\* 0: Upsloping: better heart rate with excercise (uncommon)

\* 1: Flatsloping: minimal change (typical healthy heart)

\* 2: Downslopins: signs of unhealthy heart

12. ca - number of major vessels (0-3) colored by flourosopy

\* colored vessel means the doctor can see the blood passing through

\* the more blood movement the better (no clots)

13. thal - thalium stress result

\* 1,3: normal

\* 6: fixed defect: used to be defect but ok now

\* 7: reversable defect: no proper blood movement when excercising

14. target - have disease or not (1=yes, 0=no) (= the predicted attribute)

## Preparing the tools

We're going to use pandas, Matplotlib and NumPy for data analysis and manipulation.

# 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

%matplotlib inline

# Models from Scikit-Learn

from sklearn.linear\_model import LogisticRegression

from sklearn.neighbors import KNeighborsClassifier

from sklearn.ensemble import RandomForestClassifier , VotingClassifier

from sklearn.svm import SVC

# 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 , accuracy\_score

!pwd

from google.colab import drive

drive.mount('/content/drive')

## Load data

df = pd.read\_csv("/content/drive/MyDrive/kaggle/heart-disease/heart.csv")

df.shape # (rows, columns)

## Data Exploration (exploratory data analysis or EDA)

The goal here is to find out more about the data and become a subject matter export on the dataset you're working with.

1. What question(s) are you trying to solve?

2. What kind of data do we have and how do we treat different types?

3. What's missing from the data and how do you deal with it?

4. Where are the outliers and why should you care about them?

5. How can you add, change or remove features to get more out of your data?

df.head()

df.tail()

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

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

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

df.info()

# Are there any missing values?

df.isna().sum()

df.describe()

### Heart Disease Frequency according to Sex

df.sex.value\_counts()

# Compare target column with sex column

pd.crosstab(df.target, df.sex)

# Create a plot of crosstab

pd.crosstab(df.target, df.sex).plot(kind="bar",

figsize=(10, 6),

color=["salmon", "lightblue"])

plt.title("Heart Disease Frequency for Sex")

plt.xlabel("0 = No Diesease, 1 = Disease")

plt.ylabel("Amount")

plt.legend(["Female", "Male"]);

plt.xticks(rotation=0);

### Age vs. Max Heart Rate for Heart Disease

# Create another figure

plt.figure(figsize=(10, 6))

# Scatter with postivie examples

plt.scatter(df.age[df.target==1],

df.thalach[df.target==1],

c="salmon")

# Scatter with negative examples

plt.scatter(df.age[df.target==0],

df.thalach[df.target==0],

c="lightblue")

# Add some helpful info

plt.title("Heart Disease in function of Age and Max Heart Rate")

plt.xlabel("Age")

plt.ylabel("Max Heart Rate")

plt.legend(["Disease", "No Disease"]);

# Check the distribution of the age column with a histogram

df.age.plot.hist();

### Heart Disease Frequency per Chest Pain Type

3. cp - chest pain type

\* 0: Typical angina: chest pain related decrease blood supply to the heart

\* 1: Atypical angina: chest pain not related to heart

\* 2: Non-anginal pain: typically esophageal spasms (non heart related)

\* 3: Asymptomatic: chest pain not showing signs of disease

pd.crosstab(df.cp, df.target)

# Make the crosstab more visual

pd.crosstab(df.cp, df.target).plot(kind="bar",

figsize=(10, 6),

color=["salmon", "lightblue"])

# Add some communication

plt.title("Heart Disease Frequency Per Chest Pain Type")

plt.xlabel("Chest Pain Type")

plt.ylabel("Amount")

plt.legend(["No Disease", "Disease"])

plt.xticks(rotation=0);

df.head()

# Make a correlation matrix

df.corr()

# 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)

## 5. Modelling

df.head()

# Split data into X and y

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

y = df["target"]

X

y

# 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)

X\_train

y\_train, len(y\_train)

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 3 different machine learning models:

1. Logistic Regression

2. K-Nearest Neighbours Classifier

3. Random Forest Classifier

# Put models in a dictionary

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

"KNN": KNeighborsClassifier(),

"Random Forest": RandomForestClassifier()}

# 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

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

### Model Comparison

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

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

Now we've got a baseline model... and we know a model's first predictions aren't always what we should based our next steps off. What should we do?

Let's look at the following:

\* Hypyterparameter tuning

\* Feature importance

\* Confusion matrix

\* Cross-validation

\* Precision

\* Recall

\* F1 score

\* Classification report

\* ROC curve

\* Area under the curve (AUC)

### Hyperparameter tuning (by hand)

# 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))

train\_scores

test\_scores

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}%")

## Hyperparameter tuning with RandomizedSearchCV

We're going to tune:

\* LogisticRegression()

\* RandomForestClassifier()

... using RandomizedSearchCV

# 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...

# 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)

rs\_log\_reg.best\_params\_

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

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

# 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)

# Find the best hyperparameters

rs\_rf.best\_params\_

# Evaluate the randomized search RandomForestClassifier model

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

## Evaluting our tuned machine learning classifier, beyond accuracy

\* ROC curve and AUC score

\* Confusion matrix

\* Classification report

\* Precision

\* Recall

\* F1-score

... and it would be great if cross-validation was used where possible.

To make comparisons and evaluate our trained model, first we need to make predictions.

# Make predictions with tuned model

y\_preds = rs\_rf.predict(X\_test)

y\_preds

y\_test

# Plot ROC curve and calculate and calculate AUC metric

plot\_roc\_curve(rs\_rf, X\_test, y\_test)

# Confusion matrix

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

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 + 0.5, top - 0.5)

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.

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

### 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()`.

# Check best hyperparameters

rs\_rf.best\_params\_

# Create a new classifier with best parameters

clf = LogisticRegression(C=0.20433597178569418,

solver="liblinear")

# Cross-validated accuracy

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

X,

y,

cv=5,

scoring="accuracy")

cv\_acc

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

cv\_acc

# Cross-validated precision

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

X,

y,

cv=5,

scoring="precision")

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

cv\_precision

# Cross-validated recall

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

X,

y,

cv=5,

scoring="recall")

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

cv\_recall

# Cross-validated f1-score

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

X,

y,

cv=5,

scoring="f1")

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

cv\_f1

# 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);

# Fit an instance of LogisticRegression

clf = LogisticRegression(C=0.20433597178569418,

solver="liblinear")

clf.fit(X\_train, y\_train);

# Check coef\_

clf.coef\_

df.head()

# Match coef's of features to columns

feature\_dict = dict(zip(df.columns, list(clf.coef\_[0])))

feature\_dict

# Voting Classifier

rf = RandomForestClassifier()

log\_reg = LogisticRegression(solver="lbfgs",max\_iter= 1000)

svc = SVC()

vot\_clf = VotingClassifier(

estimators=[("rf",rf),("rs",log\_reg),("svc",svc)],

voting = "hard"

)

vot\_clf.fit(X\_train, y\_train)

for clf in (log\_reg, rf, svc, vot\_clf):

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

y\_pred = clf.predict(X\_test)

print(clf.\_\_class\_\_.\_\_name\_\_, accuracy\_score(y\_test, y\_pred))