Heart Disease Classification

Description:

- Implemented machine learning classification algorithms namely KNN, logistic regression, naïve bayes, k-means, random forest, and decision tree.
- Compared the performance of models using metrics such as Accuracy, Precision, Recall, F1 score, ROC, and AUC in python.

Table of Contents

Dataset and Source

This dataset has been consolidated by Centers for Disease Control and Prevention (CDC) to study various factors leading to heart related problems which are leading cause of death for people in various countries. I have obtained the data set from Kaggle website.

Problem to be solved

The goal is to decode the relationship between different conditions and heart disease and find hidden patterns in the dataset, to uncover the correlation between the attributes causing them. Thus, we have used various predictive models so that we can accurately predict heart disease in other individuals.

For instance, if a new person comes visits hospital or a clinic. They would need the attributes mentioned above from the person. Medical representative would need to feed them into our model, and it would tell them whether, this individual has or will have heart related problem. If diagnosed so, the person in question can work on preventive measures under the guidance of medical representative to prevent heart problems before it becomes fatal.

Rows and Columns

The dataset has 319,796 rows, in other words the data belongs to 319,796 patients. There are 18 columns in this dataset, which means that there are 18 attributes quantifying the health parameters of the patients.

Data Types

Most of the datatypes are “object”, 14 attributes in total. Followed by 4 attributes of “float64” type as shown in Fig. 1.

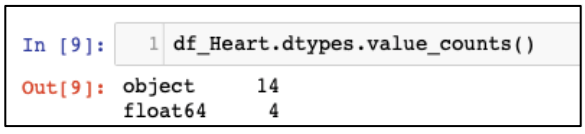

Fig. 1: Data Types

The “object” datatypes are categorical in nature. During data pre-processing stage we will convert them either using One-hot encoding or Label encoding based on the nature of data.

Data Dictionary

The dataset has 18 columns, below table (Table 1) is the dictionary to understand their meaning.

Table 1: Data Dictionary

S.No.	Column Name	Datatype	Data property	Description
1	HeartDisease	object	Categorical	This is the target column, categorical in nature, the one our model is going to predict. Possible values are No or Yes. Yes means that patient has reported coronary heart disease (CHD) or myocardial infarction (MI).
2	BMI	float64	Continuous	Continuous variable containing body mass index value.
3	Smoking	object	Categorical	Whether patient smokes or not.
4	AlcoholDrinking	object	Categorical	Whether patient drinks alcohol or not.
5	Stroke	object	Categorical	Patient ever had a stroke
6	PhysicalHealth	float64	Continuous	How many days of injury in last 30 days?
7	MentalHealth	float64	Continuous	How many days you were not mentally healthy in last 30 days?
8	DiffWalking	object	Categorical	Do you have serious difficulty walking or climbing stairs?
9	Sex	object	Categorical	Gender of patient
10	AgeCategory	object	Categorical	Age bucket
11	Race	object	Categorical	Race of patient
12	Diabetic	object	Categorical	Is patient diabetic?
13	PhysicalActivity	object	Categorical	How many days of physical activity in last 30 days?
14	GenHealth	object	Categorical	How patient feels his general health?
15	SleepTime	float64	Continuous	Hours of sleep in 24 hours?
16	Asthma	object	Categorical	Has asthama?
17	KidneyDisease	object	Categorical	Has kidney related disease?
18	SkinCancer	object	Categorical	Has skin cancer?

Libraries such as numpy, pandas, seaborn, matplotlib.pyplot, etc. has been installed and univariate and multivariate data analysis has been done in Python 3 environment as mentioned in previous projects.

Using the shape function we can see that there are 319795 rows and 18 columns in our dataset (Fig. 2).

```
df_Heart.shape
Out[8]: (319795, 18)
```

Fig. 2: Rows and columns in Dataset

Using the info() function we are able to distinguish the data types for each variable if it's an object or a float (Fig. 3). Most of the datatypes are "object", 14 attributes in total. Followed by 4 attributes of "float64".

Using the info() function we are able to distinguish the data types for each variable if it's an object or a float (Fig. 3). Most of the datatypes are "object", 14 attributes in total. Followed by 4 attributes of "float64".

```
df_Heart.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 319795 entries, 0 to 319794
Data columns (total 18 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   HeartDisease      319795 non-null  object
 1   BMI               319795 non-null  float64
 2   Smoking           319795 non-null  object
 3   AlcoholDrinking   319795 non-null  object
 4   Stroke            319795 non-null  object
 5   PhysicalHealth    319795 non-null  float64
 6   MentalHealth      319795 non-null  float64
 7   DiffWalking       319795 non-null  object
 8   Sex               319795 non-null  object
 9   AgeCategory       319795 non-null  object
 10  Race              319795 non-null  object
 11  Diabetic          319795 non-null  object
 12  PhysicalActivity  319795 non-null  object
 13  GenHealth         319795 non-null  object
 14  SleepTime         319795 non-null  float64
 15  Asthma            319795 non-null  object
 16  KidneyDisease     319795 non-null  object
 17  SkinCancer        319795 non-null  object
dtypes: float64(4), object(14)
memory usage: 43.9+ MB
```

Fig. 3: Rows and columns in Dataset

Using the correlation map we could conclude that Smoking, Stroke, PhysicalHealth, DiffWalking, AgeCategory, Diabetic, PhysicalActivity and KidneyDisease are the most significant columns. They are affecting the target column "HeartDisease" the most.

Machine Learning Models

Model 1 - Random Forest

Our dataset is a classification problem; hence I have used Random Forest model for predicting the heart disease with given indicators. A Random Forest is an ensemble technique capable of performing both regression and classification tasks with the use of multiple decision trees and a bagging technique. It combines multiple decision trees and determine the final output rather than relying on individual decision trees.

First, I have separated the columns into dependent and independent variables (or features and labels). Features are 'BMI', 'Smoking', 'AlcoholDrinking', 'Stroke', 'PhysicalHealth', 'MentalHealth', 'DiffWalking', 'Sex', 'AgeCategory', 'PhysicalActivity', 'SleepTime', 'Asthma', 'KidneyDisease', 'SkinCancer', 'Race', 'Diabetic', 'GenHealth' and Label is HeartDisease. Then I have split those variables into training set (80%) and testing set (20%). As seen in below Fig. 4, 0 depicts a person is not having heart disease and 1 depicts presence of heart disease. Oversampling has been done to make the data balanced.

```
Data in Train:
0    233851
1    233851
Name: HeartDisease, dtype: int64
```

```
Data in Test:
0    58571
1    58571
Name: HeartDisease, dtype: int64
```

Fig. 4: Train and Test data - Balanced

Random Forest Classification Report

Finally, I have imported RandomForestClassifier and fitted the data. I have trained the model on the training set and perform predictions on the test set.

```
model: RandomForestClassifier()
Accuracy_score: 0.7778514234399889
Precission_score: 0.7575542686049719
Recall_score: 0.819475815043682
F1-score: 0.7872993776613167
```

Fig. 5: RF Classification report

Accuracy for our model is 77.7% so our model is 77% accurate to predict if a person will have heart disease. Precision is a measure for the correctness of a positive prediction of a person having heart disease which is 75% and Recall is the measure for how many true positives get predicted out of all the positives in the dataset, which is 81%. F1 score combines precision and recall into one metric.

Confusion Matrix

We also have confusion matrix on held out data, training set.

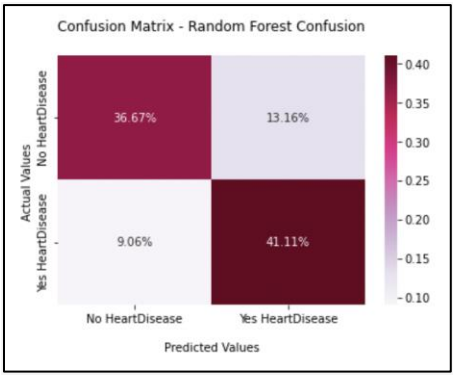

Fig. 6: Confusion Matrix plot

As we see in Fig. 41.11 % of patients with heart disease were correctly classified and 36.67% were incorrectly classified. These plots can be used to diagnose our model and decide whether it's doing well enough to put into production.

ROC Curve

A ROC Curve is a performance measurement for a classification problem at various thresholds settings. This curve displaying the percentage of true positives predicted by the model as the prediction probability cut-off is lowered from 1 to 0. AUC or the Area under curve is an indication of how well the model can separate the classes. It can range from 0.5 to 1, and the larger it is the better. Below Fig.7 is the output of ROC curve:

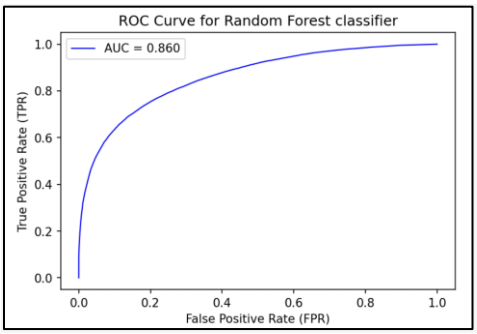

Fig. 7: ROC plot

We get a plot blue line going up to the corner and then across. So, 86% AUC that indicates the model 86.6% accurately predict whether the patient suffered from heart disease or not.

Model 2 – Logistic Regression

Logistic regression is a supervised learning classification algorithm which is used to predict the probability of a target variable. It is a process of modeling the probability of a discrete outcome by having the log-odds for the event be a linear combination of one or more independent variables.

We now build the logistic regression model using the Logistic Regression function of the sklearn library. We will set the random state parameter to 0 and then perform simple model.fit() function which will fit the model to the training instances and model.predict function which test the model to the testing instances.

```
# Create LogisticRegression
lr = LogisticRegression(random_state=0)
lr.fit(x_train, y_train)
lr_y_predict = lr.predict(x_test)
```

Fig 8: Building Logistic Regression Classifier

Now we will see the performance of our model by analyzing several metrics including accuracy score, precision score, recall score and F1 score.

```
model: LogisticRegression(random_state=0)
Accuracy_score: 0.7373771928886799
Precission_score: 0.731857012523845
Recall_score: 0.7520988706584274
F1-score: 0.7418398873452573
```

Fig. 9: Logistic Regression Classification Report

Accuracy for our model is 73.7% so our model is 73% accurate to predict if a person will have heart disease. Precision is a measure for the correctness of a positive prediction of a person having heart disease which is 73% and Recall is the measure for how many true positives get predicted out of all the positives in the dataset, which is 75%. F1 score combines precision and recall into one metric which is 74%.

We will now build a confusion matrix for the given training set. From which we see that 37.73 % of patients with heart disease were correctly classified and 36% were incorrectly classified.

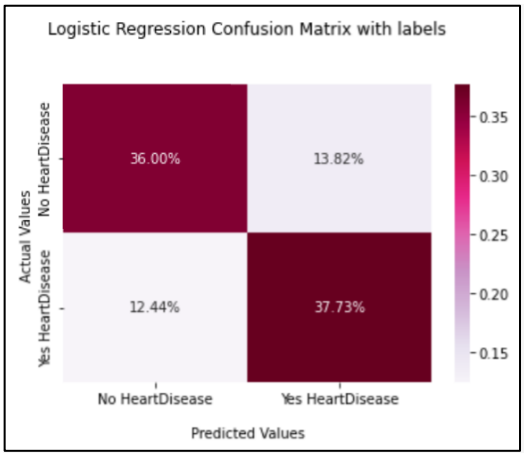

Fig. 10: Confusion Matrix

Model 2 – Logistic Regression Equation with Coefficients

Logistic Regression is basically a non-linear function defined by logit function. This logit function is a sigmoid function which has a S shaped curve (Fig. 11), the result from Logistic Regression is usually a probability value. This shape is to ensure that either the values are close to 0 or to 1.

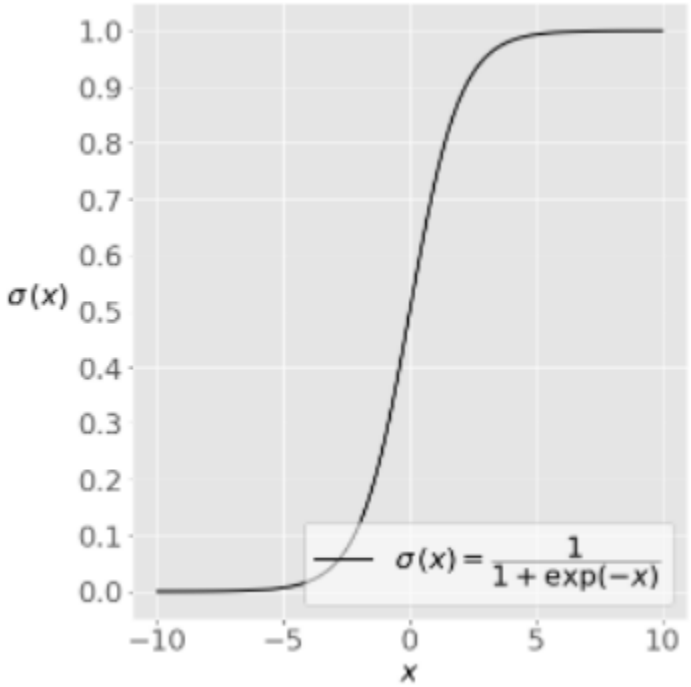

Fig. 11: S-shaped sigmoid function for a single variable X

Typical threshold is kept at 0.5, which means if answer of Logistic Regress is probability value less than 0.5 then that datapoint belongs to class 0, otherwise to class 1 (Fig. 12).

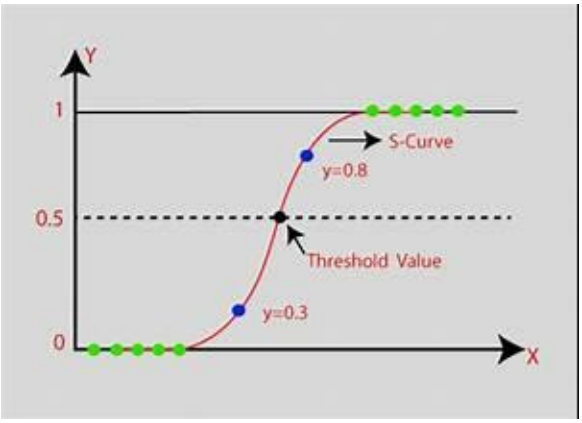

Fig. 12: Threshold value of 0.5 in Logistic Regression

As we saw in Fig. 11, sigmoid function is represented in form of exponential with linear equation. So, we found the coefficients of our Logistic Regression model, and their value is shown inf Fig. 13.

```
In [61]: print (pd.DataFrame(lr.coef_.transpose(),columns = ["Coeff Value"], index = x_train.columns))
```

```
                Coeff Value
AgeCategory        0.286551
DiffWalking        0.476026
Stroke             1.326626
Diabetic           0.367437
KidneyDisease      0.773792
PhysicalHealth     0.028188
GenHealth         -0.020383
Smoking            0.546487
```

Fig. 13: Coefficients of our Logistic Regression model

Because these coefficients are of a linear equation on exponential, therefore if is hard to quantify on how the change in these coefficients will impact the target column.

Model 3 - K Nearest Neighbor (KNN)

KNN is a classification algorithm that works by finding the distances between a query and all the examples in the data, selecting the specified number examples (K) closest to the query, then votes for the most frequent label (in the case of classification) or averages the labels (in the case of regression).

We now build the K Nearest Neighbor model using the KNeighborsClassifier() function of the sklearn library. We will select the number of estimators parameter to 100 and then perform simple model.fit() function which will fit the model to the training instances and model.predict function which test the model to the testing instances.

```
# Create KNeighborsClassifier
k_classifier = KNeighborsClassifier()
k_classifier.fit(x_train, y_train)
k_y_predict = k_classifier.predict(x_test)
```

Fig. 14: Building K Nearest Neighbor Classifier

Now we will see the performance of our model by analyzing several metrics including accuracy score, precision score, recall score and F1 score.

```
model: KNeighborsClassifier()
Accuracy_score: 0.7403063897114635
Precission_score: 0.7376942461150777
Recall_score: 0.7485403792883017
F1-score: 0.7430777366472766
```

Fig. 15: K Nearest Neighbor Classification Report

Accuracy for our model is 75.03% so our model is 73% accurate to predict if a person will have heart disease. Precision is a measure for the correctness of a positive prediction of a person having heart disease which is 73% and Recall is the measure for how many true positives get predicted out of all the positives in the dataset, which is 74%. F1 score combines precision and recall into one metric which is 74%.

We will now build a confusion matrix for the given training set. From which we see that 37.55 % of patients with heart disease were correctly classified and 36.48% were incorrectly classified.

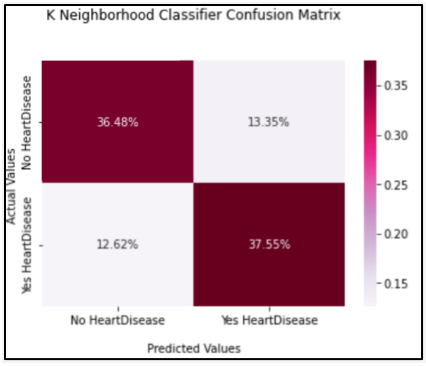

Fig. 16: Confusion Matrix

Model 4 – Naïve Bayes

Naïve Bayes is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions. It is a probabilistic classifier, which means it predicts based on the probability of an object.

We now build the naïve's model using the GaussianNB() function of the sklearn library. We use the simple model.fit() function which will fit the model to the training instances and model.predict function which test the model to the testing instances.

```
#Create Naive Bayes
clf3 = GaussianNB()
clf3.fit(x_train, y_train)
y_pred3 = clf3.predict(x_test)
```

Fig. 17: Building Naïve Bayes Classifier

Now we will see the performance of our model by analyzing several metrics including accuracy score, precision score, recall score and F1 score.

```
model: GaussianNB()
Accuracy_score: 0.7403063897114635
Precission_score: 0.7376942461150777
Recall_score: 0.7485403792883017
F1-score: 0.7430777366472766
```

Fig. 18: Naïve Bayes Classification Report

Accuracy for our model is 74.03% so our model is 74% accurate to predict if a person will have heart disease. Precision is a measure for the correctness of a positive prediction of a person having heart disease which is 73.7% and Recall is the measure for how many true positives get predicted out of all the positives in the dataset, which is 74.8%. F1 score combines precision and recall into one metric which is 74.3%.

We will now build a confusion matrix for the given training set. From which we see that 28.78 % of patients with heart disease were correctly classified and 41.16% were incorrectly classified.

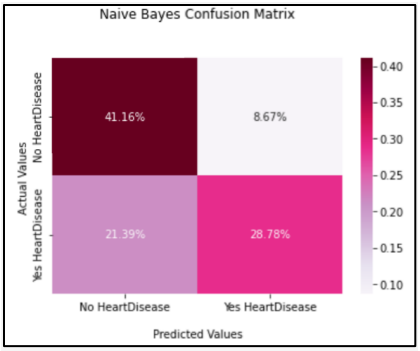

Fig. 19: Confusion Matrix

Model 5 – Decision Tree

The goal of the decision tree algorithm is to create a model that predicts value of a target variable, for which decision tree uses the tree representation to solve the problem in which the leaf node corresponds to a class label and attributes are represented on internal node of the tree.

We now build the decision tree model using the DecisionTreeClassifier function of the sklearn library. We will select the criterion parameter as entropy and then perform simple model.fit() function which will fit the model to the training instances and model.predict function which test the model to the testing instances.

```
#Create Decision Tree
dt = DecisionTreeClassifier(criterion='entropy')
dt.fit(x_train, y_train)
y_pred5 = dt.predict(x_test)
```

Fig. 20: Building Decision Tree Classifier

Now we will see the performance of our model by analyzing several metrics including accuracy score, precision score, recall score and F1 score.

```
model: DecisionTreeClassifier(criterion='entropy')
Accuracy_score: 0.7776376134529244
Precission_score: 0.7574993101817179
Recall_score: 0.8189644150862987
F1-score: 0.7870336241143465
```

Fig. 21: Decision Tree Classification Report

Accuracy for our model is 77.7% so our model is 77% accurate to predict if a person will have heart disease. Precision is a measure for the correctness of a positive prediction of a person having heart disease which is 75% and Recall is the measure for how many true positives get predicted out of all the positives in the dataset, which is 81%. F1 score combines precision and recall into one metric which is 78%.

From confusion matrix for the given training set we see that 41.09 % of patients with heart disease were correctly classified and 36.68% were incorrectly classified.

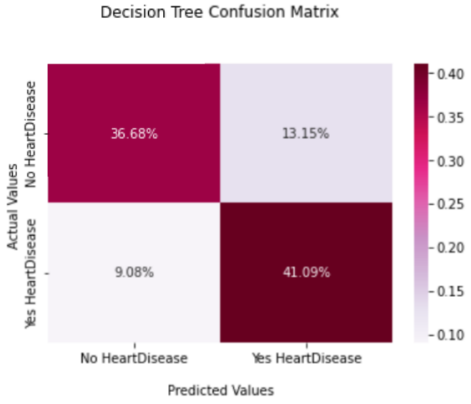

Fig. 22: Confusion Matrix

Model 5 –Decision Tree Feature Importance

One of the salient properties of Decision Tree is to identify the most important attributes of the dataset. This comes out of the box in Decision Tree, and this shows that which columns were prominently used by Decision Tree to split the data rows as it tends to expand from top to bottom, while reducing entropy at every level.

Fig. 23 shows feature importance in decreasing order. We could clearly see that the most important is AgeCategory followed by DiffWalking, so on and so forth.

```
In [50]: print (pd.DataFrame(dt.feature_importances_, columns = ["Imp"], index = x_train.columns))
```

	Imp
AgeCategory	0.443662
DiffWalking	0.112912
Stroke	0.050649
Diabetic	0.048609
KidneyDisease	0.020469
PhysicalHealth	0.138193
GenHealth	0.153717
Smoking	0.031790

Fig. 23: Feature Importance by Decision Tree

Model 5 –Decision Tree with Grid Search for Hyper Parameter Tuning

As opposed to parameters (like the ones in linear regression slope and constant term) which change based on the data for a given parametric model, hyper-parameters are preset values even before a non-parametric model gets trained on the data. The process of setting the right hyper-parameters to get max performance out of a given model, is called Hyper-parameter Tuning
Both are the two most common methods of choosing the right hyper-parameters. GridSearchCV and RandomizdSearchCV are two methods for hyper parameter tuning. They both are included in the sklearn library to perform the same over a parameter grid, that is passed as an argument to the functions along with the estimator. In Grid search, each combination of hyper-parameters tested before selecting the 'best' combination of hyper-parameters. In Random search, only a subset of combinations can be tested before selecting the 'best' combination of hyper-parameters. We use Random Search when the parameter grid is fairly large, and we want to save on processing time.
For our model, we grid criteria we used is shown in Fig. 24. After running decision tree on the g rid, the best set of parameters comes out to be 'criterion': 'gini', 'max_depth': 8 and 'min_samp les_split': 2.

```
In [60]: param_grid = {'criterion': ('entropy','gini'),
                      'max_depth': list(range(1,9)),
                       'min_samples_split':  list(range(1,5))}

In [61]: gs = GridSearchCV(dt,param_grid,cv=10)

In [63]: gs.fit(x_train, y_train)

Out[63]: GridSearchCV(cv=10, estimator=DecisionTreeClassifier(criterion='entropy'),
                      param_grid={'criterion': ('entropy', 'gini'),
                                  'max_depth': [1, 2, 3, 4, 5, 6, 7, 8],
                                  'min_samples_split': [1, 2, 3, 4]})

In [64]: gs.best_params_

Out[64]: {'criterion': 'gini', 'max_depth': 8, 'min_samples_split': 2}

In [65]: gs.cv_results_['params']

Out[65]: [{'criterion': 'entropy', 'max_depth': 1, 'min_samples_split': 1},
          {'criterion': 'entropy', 'max_depth': 1, 'min_samples_split': 2},
          {'criterion': 'entropy', 'max_depth': 1, 'min_samples_split': 3},
          {'criterion': 'entropy', 'max_depth': 1, 'min_samples_split': 4},
          {'criterion': 'entropy', 'max_depth': 2, 'min_samples_split': 1},
          {'criterion': 'entropy', 'max_depth': 2, 'min_samples_split': 2},
```

Fig. 24: Grid search criteria for Decision Tree

Now based on best set of parameters we build the decision tree model and it gave us the best accuracy as shown in Fig. 25.

```
In [67]: #Best Decision Tree on mushroom dataset
dt_Best = DecisionTreeClassifier(criterion='gini', max_depth=8, min_samples_split=2)
dt_Best.fit(x_train, y_train)
y_pred5 = dt_Best.predict(x_test)
print(f'model: {str(dt_Best)}')
print(f'Accuracy_score: {accuracy_score(y_test,y_pred5)}')
print(f'Precission_score: {precision_score(y_test,y_pred5)}')
print(f'Recall_score: {recall_score(y_test,y_pred5)}')
print(f'F1-score: {f1_score(y_test,y_pred5)}')

model: DecisionTreeClassifier(max_depth=8)
Accuracy_score: 0.7530922269379202
Precission_score: 0.7414890167787955
Recall_score: 0.77969316002557
F1-score: 0.7601113442323273
```

Fig. 25: Best decision tree on Mushroom Dataset

Constructing the ROC Curve:

Now we will use the ROC Curve for each classifier to see which one yields the best result.

```
ax = plt.gca()
knn_disp = RocCurveDisplay.from_estimator(k_classifier, x_test, y_test, ax=ax)
rdf_disp = RocCurveDisplay.from_estimator(model, x_test, y_test, ax=ax)
lg_disp = RocCurveDisplay.from_estimator(lr, x_test, y_test, ax=ax)
cl3_disp = RocCurveDisplay.from_estimator(clf3, x_test, y_test, ax=ax)
dt_disp = RocCurveDisplay.from_estimator(dt, x_test, y_test, ax=ax)
plt.show()
```

Fig. 26: Building the ROC Curve

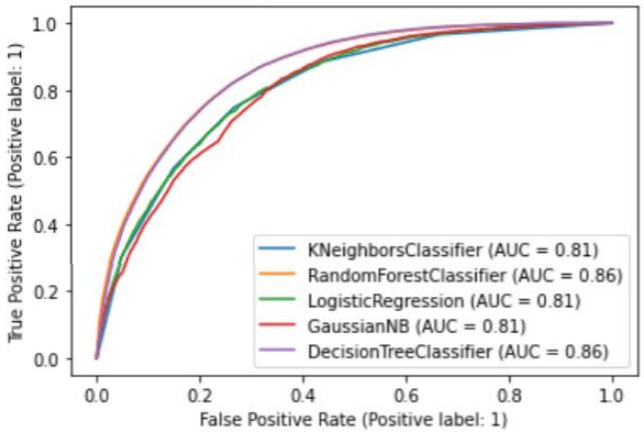

Fig. 27: ROC Curve for all models

From the ROC Curve and Area under the curve (AUC) metric, we can say that Random Forest and Decision tree classifiers works best as it gives AUC value of 0.86 and an accuracy of 77%. While KNN, Naïve Bayes and Logistic Regression yield AUC value of 0.81 & accuracy of 74%.

Model 6 – Ensemble Model (Stacking)

Our dataset is a classification problem; hence we will build Decision Tree classifier, Logistic Regression, Random Forest, Bagging Classifier and Naïve Bayes classification model on the dataset. Here, Naïve Bayes classification will set the threshold for rest of the models. All other models should have a accuracy higher than Naïve Bayes classification (Fig. 28).

```
In [121]:
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier

In [122]:
dt_model = DecisionTreeClassifier(criterion = 'entropy', max_depth = 5)
lrcl = LogisticRegression(random_state=1)
rfcl = RandomForestClassifier(random_state=1)
nbcl = GaussianNB()

# Bagging done here, base estimater is Decision tree. Hence, it is bagged decision tree
bgcl = BaggingClassifier(base_estimator=dt_model, n_estimators=10 , random_state=1)
```

Fig. 28: Machine learning models

We are using the technique of Stacking, which is a type of ensemble model. As part of stacking, all the machine learning models are clubbed together using VotingClassifier in Python. In other words, each data point will get classified against every model and final vote is taken based on their results. The class in majority is the class decided for that datapoint (Fig. 29).

```
In [123]:
enclf = VotingClassifier(estimators = [('lor', lrcl), ('rf', rfcl), ('nb', nbcl), ('bg', bgcl)], voting = 'hard')
```

Fig. 29: Ensemble model (Stacking)

Model 6 – Ensemble Model (Stacking) using KFold validation

For the assignment I have used KFold validation to have a range of accuracy score as compared to a specific value. We have kept the value of k to be 5, which means the data is divided into 5 equal parts. At each step, 4 out of 5 parts are used for training and 5th part is used for testing. Therefore, we will get 5 accuracies for every model. We used the mean value to find the mean accuracy and standard deviation to find the lower and upper end of model's accuracy (Fig. 30).

```
:
for clf, label in zip([lrcl , rfcl, nbcl, enclf, bgcl], ['Logistic Regression', 'RandomForest', 'NaiveBayes', 'Ens
    scores = cross_val_score(clf, x_train, y_train, cv=5, scoring='accuracy')
    print("Accuracy: %0.02f (+/- %0.2f) [%s]" % (scores.mean(), scores.std(), label ))
```

Fig. 30: K-Fold validation

Model 7 – K-means Clustering (Unsupervised Learning)

K-means clustering is an unsupervised machine learning algorithm which is used to find groups in the data. It uses centroids and assigns every data point to the nearest centroid. When a lot of points are near each other, you mark them as one cluster.

We now use the elbow method to find the most optimal value of k for our k-means clustering algorithm:

```
x = train_df[['AgeCategory','DiffWalking','Stroke','Diabetic','KidneyDisease','PhysicalHealth','GenHealth','Smoking']]
sse = {}
for k in range(1, 10):
    kmeans = KMeans(n_clusters=k, max_iter=1000).fit(x)
    data["clusters"] = kmeans.labels_
    sse[k] = kmeans.inertia_
plt.figure()
plt.plot(list(sse.keys()), list(sse.values()))
plt.xlabel("Number of cluster")
plt.ylabel("SSE")
plt.show()
```

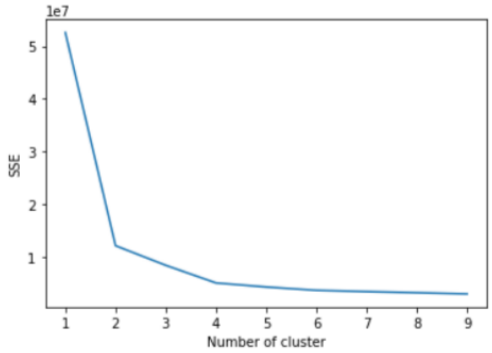

Fig. 31: Elbow method for k-means

From the elbow method we see that the most optimal value of k is 2.

```
#For K value 2
x = train_df[['AgeCategory','DiffWalking','Stroke','Diabetic','KidneyDisease','PhysicalHealth','GenHealth','Smoking']]
y = train_df['HeartDisease']
kmeans = KMeans(n_clusters=2, random_state=0)
kmeans.fit(x)
# check how many of the samples were correctly labeled
labels = kmeans.labels_
correct_labels = sum(y == labels)
print("Result: %d out of %d samples were correctly labeled." % (correct_labels, y.size))
print('Accuracy score: {0:0.2f}'. format(correct_labels/float(y.size)))
```

```
Result: 194857 out of 467702 samples were correctly labeled.
Accuracy score: 0.42
```

Fig. 32: k-mean clustering for k value 2

Now obeying to the elbow method, we built the clustering model for k value 2. We used the KMeans() function of the sklearn library seting the n_clusters argument as 2 and random state argument as 0. Then we simply use the fit() function to fit the model. We then compared the predicted labels of k means with our actual labels to check how many of the samples were correctly labeled. On doing that we observed an accuracy of 42% for the k value of 2.

```
#For K value 3
x = train_df[['AgeCategory','DiffWalking','Stroke','Diabetic','KidneyDisease','PhysicalHealth','GenHealth','Smoking']]
y = train_df['HeartDisease']
kmeans = KMeans(n_clusters=3, random_state=0)
kmeans.fit(x)
# check how many of the samples were correctly labeled
labels = kmeans.labels_
correct_labels = sum(y == labels)
print("Result: %d out of %d samples were correctly labeled." % (correct_labels, y.size))
print('Accuracy score: {0:0.2f}'. format(correct_labels/float(y.size)))
```

```
Result: 250539 out of 467702 samples were correctly labeled.
Accuracy score: 0.54
```

Fig. 33: k-mean clustering for k value 3

We now also test the same k-means clustering algorithm for k value of 3 and observed an increase in the accuracy when we compared the predicted labels with the actual labels. We observed an accuracy of 54% while performing clustering keeping the number of cluster argument as 3.

```
#For K value 4
x = train_df[['AgeCategory','DiffWalking','Stroke','Diabetic','KidneyDisease','PhysicalHealth','GenHealth','Smoking']]
y = train_df['HeartDisease']
kmeans = KMeans(n_clusters=4, random_state=0)
kmeans.fit(x)
# check how many of the samples were correctly labeled
labels = kmeans.labels_
correct_labels = sum(y == labels)
print("Result: %d out of %d samples were correctly labeled." % (correct_labels, y.size))
print('Accuracy score: {0:0.2f}'. format(correct_labels/float(y.size)))
```

```
Result: 147641 out of 467702 samples were correctly labeled.
Accuracy score: 0.32
```

```
#For K value 5
x = train_df[['AgeCategory','DiffWalking','Stroke','Diabetic','KidneyDisease','PhysicalHealth','GenHealth','Smoking']]
y = train_df['HeartDisease']
kmeans = KMeans(n_clusters=5, random_state=0)
kmeans.fit(x)
# check how many of the samples were correctly labeled
labels = kmeans.labels_
correct_labels = sum(y == labels)
print("Result: %d out of %d samples were correctly labeled." % (correct_labels, y.size))
print('Accuracy score: {0:0.2f}'. format(correct_labels/float(y.size)))
```

```
Result: 143372 out of 467702 samples were correctly labeled.
Accuracy score: 0.31
```

Fig. 34: k-mean clustering for k value 4 & 5

No significant improvement in accuracy was seen while performing k-means clustering for k values 4 and 5. In fact we see that the accuracy for k values 4 and 5 actually dropped to 30%.

Conclusion

In this project, we learnt how to implement several machine learning classification algorithms namely random forest, K nearest neighbor, logistic regression, naïve bayes and decision tree in python and how to compare all the models using the Accuracy, Precision, Recall, F1 score, ROC, and AUC (Area under the curve) in python.

Also, we tried to decode the equation of Logistic Regression by identifying the coefficients of the equation. However, it is very difficult to quantify how each attribute affects the target variable. Because unlike Linear Regression, the relationship between variables and target column is non-linear in case of Logistic Regression.

Furthermore, we used hyper parameter tuning techniques to find out best set of hyper parameters. For that we used Grid Search, and after obtaining the best set of parameters, we build our model using those parameters.

K-Fold validation was also used to find out the range and standard deviation of model accuracies. This helps in setting up confidence interval for out models, and gives us a fair idea on how model will respond to unseen datapoints in future.

On comparison of all the models based on the given performance metric we concluded that for the given dataset of heart disease to predict whether the person will have heart disease or not, the random forests and decision trees are the two classification algorithms that works the best yielding an accuracy of 77% and AUC of 86%.

References

- *CDC Works 24/7*. (2022, April 28). Centers for Disease Control and Prevention. https://www.cdc.gov/
- *Know Your Risk for Heart Disease | cdc.gov*. (2019, December 9). Centers for Disease Control and Prevention. https://www.cdc.gov/heartdisease/risk_factors.htm
- *Personal Key Indicators of Heart Disease*. (2022, February 16). Kaggle. https://www.kaggle.com/datasets/kamilpytlak/personal-key-indicators-of-heart-disease?select=heart_2020_cleaned.csv
- Will Koehrsen (Dec 27, 2017). Towards Data Science, Random Forest in Python, https://towardsdatascience.com/random-forest-in-python-24d0893d51c0
- Jason Brownlee (April 20, 2020). Machine Learning Mastery, How to Develop a Random Forest Ensemble in Python, https://machinelearningmastery.com/random-forest-ensemble-in-python/
- Logistic Regression in Python, https://realpython.com/logistic-regression-python/