Aim: Experiment to implement any one of the classification algorithm(Decision tree/Naive Bayes) /Technique using python

To do:

1. Preprocess data. Split data into train and test set
2. Build a Classification model using a function defined by the student(model to be built using the same training data).
3. Calculate metrics based on test data using an inbuilt function
4. Compare the results of all three ways of implementation. (Rapid Miner, Python Library and self-defined function)

Link of Dataset:

This data set dates from 1988 and consists of four databases: Cleveland, Hungary, Switzerland, and Long Beach V. It contains 76 attributes, including the predicted attribute, but all published experiments refer to using a subset of 14 of them. The "target" field refers to the presence of heart disease in the patient. It is integer valued 0 = no disease and 1 = disease.
https://www.kaggle.com/datasets/johnsmith88/heart-disease-dataset

Theory:

Classification is defined as the process of recognition, understanding, and grouping of objects and ideas into preset categories a.k.a "sub-populations." With the help of these pre-categorized training datasets, classification in machine learning programs leverage a wide range of algorithms to classify future datasets into respective and relevant categories.
Based on training data, the Classification algorithm is a Supervised Learning technique used to categorize new observations. In classification, a program uses the dataset or observations provided to learn how to categorize new observations into various classes or groups. For instance, 0 or 1, red or blue, yes or no, spam or not spam, etc. Targets, labels, or categories can all be used to describe classes. The Classification algorithm uses labeled input data because it is a supervised learning technique and comprises input and output information. A discrete output function (y) is transferred to an input variable in the classification process (x).
In simple words, classification is a type of pattern recognition in which classification algorithms are performed on training data to discover the same pattern in new data sets.

Types of Classification Algorithms

You can apply many different classification methods based on the dataset you are working with. It is so because the study of classification in statistics is extensive. The top five machine learning algorithms are listed below.

1. Logistic Regression

It is a supervised learning classification technique that forecasts the likelihood of a target variable. There will only be a choice between two classes. Data can be coded as either one or yes, representing success, or as 0 or no, representing failure. The dependent variable can be predicted most effectively using logistic regression. When the forecast is categorical, such as true or false, yes or no, or a 0 or 1, you can use it. A logistic regression technique can be used to determine whether or not an email is spam.

2. Naive Bayes

Naive Bayes determines whether a data point falls into a particular category. It can be used to classify phrases or words in text analysis as either falling within a predetermined classification or not.

3. K-Nearest Neighbors

It calculates the likelihood that a data point will join the groups based on which group the data points closest to it are a part of. When using k-NN for classification, you determine how to classify the data according to its nearest neighbor.

4. Decision Tree

A decision tree is an example of supervised learning. Although it can solve regression and classification problems, it excels in classification problems. Similar to a flow chart, it divides data points into two similar groups at a time, starting with the "tree trunk" and moving through the "branches" and "leaves" until the categories are more closely related to one another.

5. Random Forest Algorithm

The random forest algorithm is an extension of the Decision Tree algorithm where you first create a number of decision trees using training data and then fit your new data into one of the created ‘trees’ as a ‘random forest’. It averages the data to connect it to the nearest tree data based on the data scale. These models are great for improving the decision tree’s problem of forcing data points unnecessarily within a category.

6. Support Vector Machine

Support Vector Machine is a popular supervised machine learning technique for classification and regression problems. It goes beyond X/Y prediction by using algorithms to classify and train the data according to polarity.

Implementation of Decision tree using Python

- Importing libraries and loading the dataset

```
[1] #Import the libraries
    import numpy as np
    import pandas as pd
    import matplotlib.pyplot as plt
    import seaborn as sb
```

```
[2] #loading the dataset
    data = pd.read_csv("heart.csv")
    data
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3	0
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3	0
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3	0
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3	0
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1020	59	1	1	140	221	0	1	164	1	0.0	2	0	2	1
1021	60	1	0	125	258	0	0	141	1	2.8	1	1	3	0
1022	47	1	0	110	275	0	0	118	1	1.0	1	1	2	0
1023	50	0	0	110	254	0	0	159	0	0.0	2	0	2	1
1024	54	1	0	120	188	0	1	113	0	1.4	1	1	3	0

1025 rows × 14 columns

- Filling the null values

```
data = data.fillna(0)
data
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3	0
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3	0
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3	0
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3	0
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

- Training the model with given columns in X. Our target column is “Target”

Decision Tree

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn import tree
```

+ Code

```
[6] print(data['target'].unique())
```

```
[0 1]
```

```
[7] X=data[['age','chol','trestbps','thalach','fbs','exang','sex','slope','ca','thal','oldpeak',]]
    y=data['target']
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
[8]  clf = DecisionTreeClassifier(max_depth=3,criterion="entropy",random_state=100)
     clf.fit(X_train, y_train)
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=100)
```

```
y_pred=clf.predict(X_test)
```

```
[10] df=pd.DataFrame(y_test,y_pred)
```

- Actual data and predicted data from the trained model

```
df=pd.DataFrame({"Actual application type":y_test,'Predicted application type':y_pred})
df
```

	Actual application type	Predicted application type
527	1	1
359	1	1
447	0	0
31	1	1
621	0	0
...	...	...
832	1	0
796	1	1
644	1	1
404	0	0
842	0	0

205 rows × 2 columns

- The accuracy of Decision Tree by using python is:

```
[12] accuracy_score(y_test,y_pred)*100
```

```
80.0
```

Implementation of Naive Bayes using Python

- Training the model

```
[13] print(data['target'].unique())
```

```
[0 1]
```

```
[14] X = data[['age','chol','trestbps','thalach','fbs','exang','sex','slope','ca','thal','oldpeak']]
     y = data['target']
```

```
[15] from sklearn.model_selection import train_test_split
     X_train, X_test, y_train, y_test = train_test_split(X,y, random_state = 0)
```

- Using Gaussian Mixture method

```
[16] from sklearn.naive_bayes import GaussianNB
     clf1=GaussianNB()
     clf1.fit(X_train,y_train)
```

```
GaussianNB
GaussianNB()
```

- Actual and predicted data

```
df1=pd.DataFrame(y_test,y_pred1)
df1=pd.DataFrame({"Actual":y_test,'Predicted':y_pred1})
print(df1)
```

```
     Actual  Predicted
807       1          1
27        0          1
77        0          0
406       1          1
886       0          0
..      ...        ...
79        1          1
973       1          0
193       1          0
361       1          0
181       1          0

[257 rows x 2 columns]
```

- Accuracy after using naive bayes with python is:

```
[17] from sklearn.metrics import classification_report, accuracy_score
     y_pred1=clf1.predict(X_test)
     score=accuracy_score(y_pred1,y_test)
     print(score)
```

```
0.8210116731517509
```

Implementation using Self defined function

```
class CustomLogisticRegression:
  def __init__(self, learning_rate=0.1, n_iterations=1000, verbose=False):
    self.learning_rate = learning_rate
    self.n_iterations = n_iterations
    self.verbose = verbose
    self.theta = None

  def fit(self, X, y):
    n_samples, n_features = X.shape
    # Initialize parameters
    self.theta = np.zeros(n_features)
    # Gradient descent
    for i in range(self.n_iterations):
  # Hypothesis function
      z = np.dot(X, self.theta)
      h = self._sigmoid(z)
      # Cost function
      J = -np.mean(y*np.log(h) + (1-y)*np.log(1-h))
      # Gradient of cost function
      grad_J = np.dot(X.T, (h-y)) / n_samples
      # Update parameters
      self.theta -= self.learning_rate * grad_J
      # Print progress
      if self.verbose and i % 100 == 0:
        print(f"Iteration {i}: J = {J}")

  def predict(self, X):
    z = np.dot(X, self.theta)
    h = self._sigmoid(z)
    y_pred = (h >= 0.5).astype(int)
    return y_pred

  def evaluate(self, X, y):
    y_pred = self.predict(X)
    accuracy = np.mean(y_pred == y)
    return accuracy

  def _sigmoid(self, z):
    return 1 / (1 + np.exp(-z))
```

```
[23] model = CustomLogisticRegression(learning_rate=0.1,n_iterations=1000,verbose=False)

     model.fit(X_train,y_train)

<ipython-input-22-a499cf63daf9>:18: RuntimeWarning: divide by zero encountered in log
  J = -np.mean(y*np.log(h) + (1-y)*np.log(1-h))
<ipython-input-22-a499cf63daf9>:39: RuntimeWarning: overflow encountered in exp
  return 1 / (1 + np.exp(-z))
```

```
[26] accuracy = model.evaluate(X_test,y_test)
     print(f"accuracy = {accuracy}")

accuracy = 0.6147859922178989
<ipython-input-22-a499cf63daf9>:39: RuntimeWarning: overflow encountered in exp
  return 1 / (1 + np.exp(-z))
```

Comparison of different models:-

Method	Naive Bayes	Decision Tree
Rapid Miner	81.49%	87.66%
Python	82.10%	80.0%
Self Defined Function	61.47%	

- The performance of Decision Tree is better than Naive Bayes in RapidMiner.
- The performance of Python is less than Rapid Miner for decision trees.
- The performance of Python is better than Rapid Miner for Naive Bayes.
- Self defined Function gave the most poor performance.

Conclusion: In this Experiment we saw two classification algorithms that are Decision Tree and Naive Bayes. We found their accuracies based on the provided dataset of heart disease prediction and compared the results for different methods. We have successfully implemented classification models like Decision Tree and Naive Bayes using python and self defined functions.