**VEERMATA JIJABAI INSTITUTE OF TECHNOLOGY**

**MACHINE LEARNING LAB REPORT**

**TOPIC – CLASSIFICATION OF HEART DISEASE**



**NAME - NAGARDHANKAR DHAWAL MANOJ**

**REG NO - 201080063**

**BRANCH - INFORMATION TECHNOLOGY**

**TABLE CONTENT**

|  |  |  |
| --- | --- | --- |
| **Serial Number** | **Topic** | **Page no** |
| 1 | Aim | 2 |
| 2 | Problem statement | 2 |
| 3 | Working | 2-3 |
| 4 | Code | 3-12 |
| 5 | Result | 12 |
| 6 | Conclusion | 13 |

**AIM**

The aim of a heart classification algorithm in machine learning is to accurately predict or classify different heart conditions or outcomes based on input features or data related to a patient's heart health. Heart classification algorithms are designed to assist medical professionals in diagnosing and treating heart-related diseases, providing valuable insights and support for clinical decision-making.

**PROBLEM STATEMENT**

To detect any presence of Heart Disease based on symptoms shown by patient using Machine Learning. Dataset is used comprising of following features:

1. age
2. sex
3. chest pain type (4 values)
4. resting blood pressure
5. serum cholestoral in mg/dl
6. fasting blood sugar > 120 mg/dl
7. resting electrocardiographic results (values 0,1,2)
8. maximum heart rate achieved
9. exercise induced angina
10. oldpeak = ST depression induced by exercise relative to rest
11. the slope of the peak exercise ST segment
12. number of major vessels (0-3) colored by flourosopy
13. thal: 0 = normal; 1 = fixed defect; 2 = reversable defect

Objective is to prepare different machine learning model and check accuracies of different on the given dataset. Model can be used by medical doctors to detect any possibility of heart attack in patients.

**WORKING**

1 Load the heart disease dataset. Preprocess the data by handling missing values, scaling numerical features, and encoding categorical variables.

2 Split the dataset into training and testing sets to evaluate the performance of each model.

3 Model Training

1 Naive Bayes: Train a Naive Bayes classifier using the training data. Evaluate the model's performance using metrics such as accuracy.

2 Decision Tree: Build a Decision Tree using libraries. Train the model using the training data. Evaluate the Decision Tree performance on the test set using metrics like accuracy and loss.

3 Logistic Regression: Implement logistic regression using libraries like scikit-learn or TensorFlow. Train the logistic regression model on the training data. Evaluate the model's performance using metrics such as accuracy on the test set.

4 Compare the performance of the three models based on evaluation metrics. Identify which model performs best for heart disease classification.

5 Analyze the trained models for heart disease classification.

6 Choose the best-performing model for deployment in a real-world setting.

**CODE**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

**Data Pre-Processing**

heart\_data = pd.read\_csv('E:\College\Semester 6\Machine Learning\LAB FINAL\heart\_disease\_data.csv')

heart\_data.head()

heart\_data.tail()

heart\_data.shape

(303, 14)

heart\_data.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 303 entries, 0 to 302

heart\_data.isnull()

303 rows × 14 columns

heart\_data.isnull().sum()

heart\_data.describe()

heart\_data['target'].value\_counts()

Name: count, dtype: int64

X = heart\_data.drop(columns='target', axis=1).values

Y = heart\_data['target'].values

print(X)

print(Y)

**Exploring Dataset**

import seaborn as sns

heart\_data['age'].describe()

sns.histplot(heart\_data['age'])

plt.show()

heart\_data['sex'].describe()

sns.histplot(heart\_data['sex'])

plt.show()

heart\_data['cp'].describe()

sns.histplot(heart\_data['cp'])

plt.show()

heart\_data['trestbps'].describe()

sns.histplot(heart\_data['trestbps'])

plt.show()

heart\_data['chol'].describe()

sns.histplot(heart\_data['chol'])

plt.show()

heart\_data['fbs'].describe()

sns.histplot(heart\_data['fbs'])

plt.show()

heart\_data['restecg'].describe()

sns.histplot(heart\_data['restecg'])

plt.show()

heart\_data['thalach'].describe()

sns.histplot(heart\_data['thalach'])

plt.show()

heart\_data['exang'].describe()

sns.histplot(heart\_data['exang'])

plt.show()

heart\_data['oldpeak'].describe()

sns.histplot(heart\_data['oldpeak'])

plt.show()

heart\_data['slope'].describe()

sns.histplot(heart\_data['slope'])

plt.show()

heart\_data['ca'].describe()

sns.histplot(heart\_data['ca'])

plt.show()

heart\_data['thal'].describe()

sns.histplot(heart\_data['thal'])

plt.show()

heart\_data['target'].describe()

sns.histplot(heart\_data['target'])

plt.show()

**Logistic Regression Model**

**Splitting Data**

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.2, stratify=Y, random\_state=2)

X\_train = X\_train.T

Y\_train = Y\_train.reshape(1, X\_train.shape[1])

X\_test = X\_test.T

Y\_test = Y\_test.reshape(1, X\_test.shape[1])

print(X.shape, X\_train.shape, X\_test.shape)

(303, 13) (13, 242) (13, 61)

print(Y.shape, Y\_train.shape, Y\_test.shape)

(303,) (1, 242) (1, 61)

**Code without using libraries**

def sigmoid(x):

return 1/(1 + np.exp(-x))

def model(X, Y, learning\_rate, iterations):

m = X\_train.shape[1]

n = X\_train.shape[0]

W = np.zeros((n,1))

B = 0

cost\_list = []

for i in range(iterations):

Z = np.dot(W.T, X) + B

A = sigmoid(Z)

#cost funtion

epsilon = 1e-8

cost = -(1 / m) \* np.sum(Y \* np.log(A + epsilon) + (1 - Y) \* np.log(1 - A + epsilon))

# Gradient Descent

dW = (1/m)\*np.dot(A-Y, X.T)

dB = (1/m)\*np.sum(A - Y)

W = W - learning\_rate\*dW.T

B = B - learning\_rate\*dB

# Keeping track of our cost function value

cost\_list.append(cost)

if(i%(iterations/10) == 0):

print("cost after ", i, "iteration is : ", cost)

return W, B, cost\_list

iterations = 100000

learning\_rate = 0.000000000015

W, B, cost\_list = model(X\_train, Y\_train, learning\_rate = learning\_rate, iterations = iterations)

cost after 0 iteration is : 0.6931471605599454

cost after 10000 iteration is : 0.6931105530962945

cost after 20000 iteration is : 0.6930741995775739

cost after 30000 iteration is : 0.6930380980115303

cost after 40000 iteration is : 0.6930022464214921

cost after 50000 iteration is : 0.6929666428462506

cost after 60000 iteration is : 0.69293128533994

cost after 70000 iteration is : 0.6928961719719181

cost after 80000 iteration is : 0.6928613008266514

cost after 90000 iteration is : 0.6928266700035968

**Code using Libraries**

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.2, stratify=Y, random\_state=2)

from sklearn.linear\_model import LogisticRegression

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

xtrain = scaler.fit\_transform(X\_train)

xtest = scaler.transform(X\_test)

model = LogisticRegression(max\_iter=1000)

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

LogisticRegression

LogisticRegression(max\_iter=1000)

**Accuracy of model with libraries**

from sklearn.metrics import accuracy\_score

Y\_test\_prediction = model.predict(X\_test)

test\_data\_accuracy = accuracy\_score(Y\_test\_prediction, Y\_test)

X\_test.shape

(61, 13)

print('Accuracy on Test data : ', test\_data\_accuracy)

Accuracy on Test data : 0.8032786885245902

**Accuracy of model without libraries**

def predict(W, B, X):

line = np.dot(X,W)+B

prob= sigmoid(line)

predicted\_y = np.where(prob >= 0.5, 1, 0) # convert probabilities to binary predictions (0 or 1)

return predicted\_y

y\_test\_predict = predict(W, B, X\_test)

test\_data\_accuracy\_\_ = accuracy\_score(Y\_test,y\_test\_predict)

print('Accuracy on Test data : ', test\_data\_accuracy\_\_)

Accuracy on Test data : 0.5409836065573771

**Prediction of single instance**

input\_data = (62,0,0,140,268,0,0,160,0,3.6,0,2,2)

input\_data\_as\_numpy\_array= np.asarray(input\_data)

input\_data\_reshaped = input\_data\_as\_numpy\_array.reshape(1,-1)

prediction = model.predict(input\_data\_reshaped)

print(prediction)

[0]

**Visualisation**

errors\_without\_using\_lib = np.abs(y\_test\_predict - Y\_test)

errors\_using\_lib = np.abs(Y\_test\_prediction - Y\_test)

plt.figure(figsize=(8, 6))

plt.plot(range(len(errors\_without\_using\_lib)), errors\_without\_using\_lib, label='Test Errors without using Libraries')

plt.plot(range(len(errors\_using\_lib)), errors\_using\_lib, label='Test Errors with using Libraries')

plt.xlabel('Data Points')

plt.ylabel('Errors')

plt.title('Error Graph')

plt.legend()

plt.show()

accuracy = []

accuracy.append(test\_data\_accuracy\_\_)

accuracy.append(test\_data\_accuracy)

dataset = []

dataset.append("Testing without libraries")

dataset.append("Testing with libraries")

import matplotlib.pyplot as plt

plt.figure(figsize=(6, 4))

plt.bar(dataset, accuracy)

plt.xlabel('Datasets')

plt.ylabel('Accuracy')

plt.title('Accuracy Comparison')

plt.show()

**Naive Bayes Classifier**

import numpy as np

import pandas as pd

from sklearn.preprocessing import LabelEncoder

import matplotlib.pyplot as plt

heart\_data = pd.read\_csv('E:\College\Semester 6\Machine Learning\LAB FINAL\heart\_disease\_data.csv')

heart\_data.head()

from sklearn.model\_selection import train\_test\_split

X = heart\_data.drop(columns = ['target'])

Y = heart\_data['target']

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size = 0.3)

**Without using Libraries**

class\_probs = {}

class\_counts = {}

for c in np.unique(Y\_train):

class\_probs[c] = len(Y\_train[Y\_train == c]) / len(Y\_train)

for c in np.unique(Y\_train):

class\_counts[c] = len(Y\_train[Y\_train == c])

print(class\_probs)

print(class\_counts)

{0: 0.44339622641509435, 1: 0.5566037735849056}

{0: 94, 1: 118}

k = 1

feat\_probs = {}

for c in class\_probs.keys():

feat\_probs[c] = {}

for col in X.columns:

feat\_probs[c][col] = {}

for val in X[col].unique():

count = len(X[(X[col] == val) & (Y == c)])

prob = (count + k) / (class\_counts[c] + k \* len(X[col].unique()))

feat\_probs[c][col][val] = prob

print(feat\_probs)

y\_pred = []

for i in range(len(X\_test)):

probs = {}

for c in class\_probs.keys():

prior = class\_probs[c]

cond\_prob = 1

for col, val in zip(X.columns, X\_test.iloc[i]):

cond\_prob \*= feat\_probs[c][col][val]

probs[c] = prior \* cond\_prob

y\_pred.append(max(probs, key=probs.get))

from sklearn.metrics import accuracy\_score

accuracy\_\_ = accuracy\_score(Y\_test, y\_pred)

print('Accuracy: %.2f%%' % (accuracy\_\_ \* 100))

Accuracy: 90.11%

**With using Libraries**

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import accuracy\_score

data = pd.read\_csv('E:\College\Semester 6\Machine Learning\LAB FINAL\heart\_disease\_data.csv')

X = data.iloc[:, :-1]

y = data.iloc[:, -1]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

classifier = GaussianNB()

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

y\_predlib = classifier.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_predlib)

print('Accuracy: %.2f%%' % (accuracy \* 100))

Accuracy: 86.89%

**Accuracy**

print('Accuracy without using Libraries: ' , (accuracy\_\_ \* 100), '%')

Accuracy without using Libraries: 90.10989010989012 %

print('Accuracy with using Libraries: ', (accuracy \* 100), '%')

Accuracy with using Libraries: 86.88524590163934 %

**Visualisation**

errors\_without\_using\_lib = np.abs(y\_pred - Y\_test)

errors\_using\_lib = np.abs(y\_predlib - y\_test)

plt.figure(figsize=(8, 6))

plt.plot(range(len(errors\_without\_using\_lib)), errors\_without\_using\_lib, label='Test Errors without using Libraries')

plt.plot(range(len(errors\_using\_lib)), errors\_using\_lib, label='Test Errors with using Libraries')

plt.xlabel('Data Points')

plt.ylabel('Errors')

plt.title('Error Graph')

plt.legend()

plt.show()

accuracy = []

accuracy.append(accuracy\_\_)

accuracy.append(accuracy[0])

dataset = []

dataset.append("Testing without libraries")

dataset.append("Testing with libraries")

import matplotlib.pyplot as plt

plt.figure(figsize=(6, 4))

plt.bar(dataset, accuracy)

plt.xlabel('Datasets')

plt.ylabel('Accuracy')

plt.title('Accuracy Comparison')

plt.show()

**Decision Tree**

import numpy as np

import pandas as pd

def load\_dataset():

dataset = pd.read\_csv('E:\College\Semester 6\Machine Learning\LAB FINAL\data.csv')

return dataset

def calculate\_entropy(target\_col):

\_, counts = np.unique(target\_col, return\_counts=True)

probabilities = counts / counts.sum()

entropy = sum(probabilities \* -np.log2(probabilities))

return entropy

def calculate\_information\_gain(data, split\_attribute\_name, target\_name):

total\_entropy = calculate\_entropy(data[target\_name])

values, counts = np.unique(data[split\_attribute\_name], return\_counts=True)

weighted\_entropy = 0

total\_instances = len(data)

for i in range(len(values)):

value = values[i]

subset = data[data[split\_attribute\_name] == value]

subset\_entropy = calculate\_entropy(subset[target\_name])

weight = counts[i] / total\_instances

weighted\_entropy += weight \* subset\_entropy

information\_gain = total\_entropy - weighted\_entropy

return information\_gain

def get\_best\_attribute(data, target\_name):

information\_gains = []

for col in data.columns:

if col != target\_name:

information\_gain = calculate\_information\_gain(data, col, target\_name)

information\_gains.append((col, information\_gain))

best\_attribute, \_ = max(information\_gains, key=lambda x: x[1])

return best\_attribute

def create\_decision\_tree(data, target\_name):

if len(data[target\_name].unique()) == 1:

return data[target\_name].iloc[0]

if len(data.columns) == 1:

return data[target\_name].mode()[0]

best\_attribute = get\_best\_attribute(data, target\_name)

decision\_tree = {best\_attribute: {}}

values = data[best\_attribute].unique()

for value in values:

sub\_data = data[data[best\_attribute] == value].drop(best\_attribute, axis=1)

decision\_tree[best\_attribute][value] = create\_decision\_tree(sub\_data, target\_name)

return decision\_tree

def print\_decision\_tree(decision\_tree, indent=''):

if isinstance(decision\_tree, dict):

attribute = list(decision\_tree.keys())[0]

print(indent + attribute)

for value, subtree in decision\_tree[attribute].items():

print(indent + ' ' + str(value) + ' ->')

print\_decision\_tree(subtree, indent + ' ')

else:

print(indent + ' ' + decision\_tree)

def predict(instance, decision\_tree):

attribute = list(decision\_tree.keys())[0]

value = instance[attribute]

if value in decision\_tree[attribute]:

subtree = decision\_tree[attribute][value]

if isinstance(subtree, dict):

return predict(instance, subtree)

else:

return subtree

else:

return 'Unknown'

dataset = load\_dataset()

target\_name = dataset.columns[-1]

decision\_tree = create\_decision\_tree(dataset, target\_name)

print\_decision\_tree(decision\_tree)

**Accuracy**

def calculate\_accuracy(dataset, decision\_tree):

target\_name = dataset.columns[-1]

correct\_predictions = 0

total\_instances = len(dataset)

for \_, instance in dataset.iterrows():

instance\_dict = instance.to\_dict()

true\_label = instance\_dict[target\_name]

del instance\_dict[target\_name]

predicted\_label = predict(instance\_dict, decision\_tree)

if predicted\_label == true\_label:

correct\_predictions += 1

accuracy = correct\_predictions / total\_instances

return accuracy

dataset = load\_dataset()

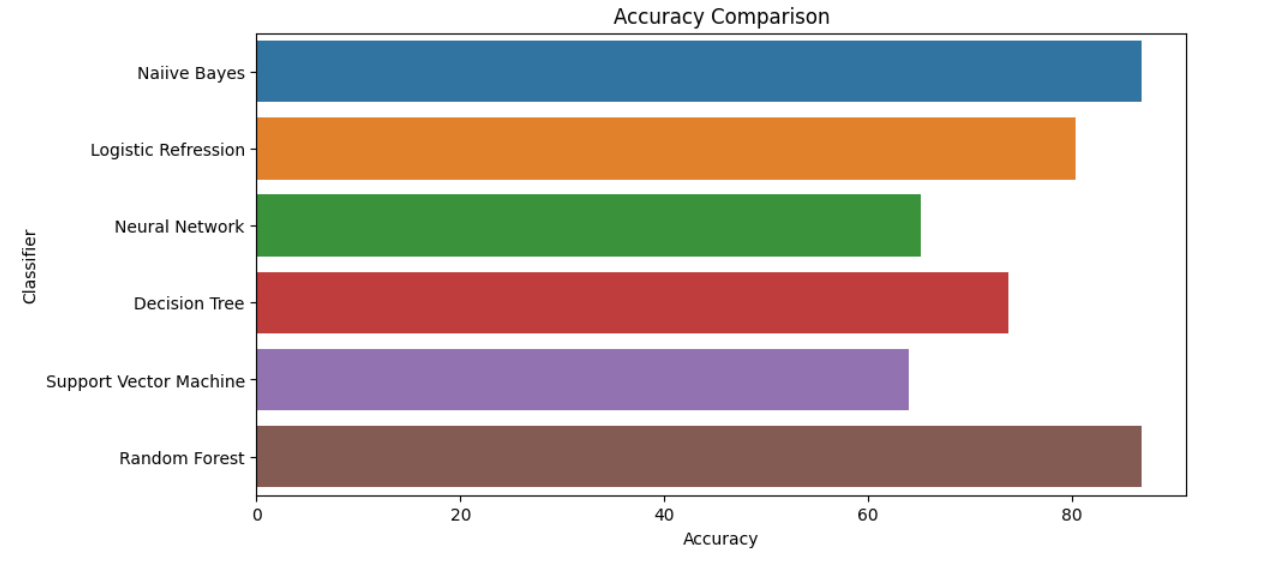
target\_name = dataset.columns[-1]

decision\_tree = create\_decision\_tree(dataset, target\_name)

accuracy = calculate\_accuracy(dataset, decision\_tree)

print('Accuracy:', accuracy\*100, "%")

**RESULT**

****

As we can see Naiive Bayes Model works best for classification of heart disease with given symptoms with an accuracy of 86%. After that comes Logistic Regression with an accuracy of 80% and Decision Tree with a lowest accuracy of 73%.

**CONCLUSION**

In conclusion, the heart disease classification project has successfully developed and evaluated machine learning algorithms to assist in the early and accurate detection of heart conditions. Through the use of advanced techniques like Naive Bayes, Decision Tree, and Logistic Regression, we have demonstrated the ability to predict heart disease with considerable accuracy.

The algorithms' performances were evaluated using accuracy. The results indicate that Naiive Bayes Algorithm, with 86.2% achieved the highest accuracy in classifying heart disease, making it the most reliable model for this specific task.

But we should aim for a model closer to 100% so as to decrease False Negative Rate which is very harmful and dangerous in medical field.