**Pneumococcus-Disease-Prediction-using-Machine-Learning**

Detecting Pneumococcus Disease is a crucial task within the medical field, given its significant impact on patient outcomes when identified early. Machine learning (ML) techniques offer promising avenues for aiding in the diagnosis of this disease. This research presents an ML model designed to leverage a dataset containing patient symptoms and laboratory test results for pneumococcus detection. The model employs feature selection techniques to identify the most relevant features, enhancing its predictive capability. Utilizing a classification algorithm, the model predicts the presence of the disease with high accuracy. The proposed model is trained using the Inception V3 algorithm and validated using a random test dataset. Evaluation of the model’s performance is conducted using the accuracy metric, re vealing a maximum detection accuracy of 96.2 percentage. These results underscore the model’s remarkable accuracy and its potential as an effective diagnostic tool for pneumococcus Disease. Medical professionals stand to benefit significantly from the implementation of this model, as it can aid in the timely and accurate diagnosis of the disease, ultimately leading to improved patient outcomes. By leveraging ML techniques, healthcare providers can enhance their diagnostic capabilities, providing more effective treatment strategies and improving overall patient care.

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

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

from sklearn.metrics import accuracy\_score

from sklearn.metrics import classification\_report

import re

import string

data\_fake = pd.read\_csv(’/content/drive/MyDrive/Colab

Notebooks/Fake.csv’)

data\_true = pd.read\_csv(’/content/drive/MyDrive/Colab

Notebooks/True.csv’)

data\_true.head()

data\_fake.head()

data\_fake["class"] = 0

data\_true[’class’] = 1

data\_fake.shape, data\_true.shape

data\_fake\_manual\_testing = data\_fake.tail(10)

for i in range(23480,23470,-1):

data\_fake.drop([i], axis = 0, inplace = True)

data\_true\_manual\_testing = data\_true.tail(10)

for i in range(21416,21406,-1):

data\_true.drop([i], axis = 0, inplace = True)

data\_fake.shape, data\_true.shape

data\_fake\_manual\_testing[’class’] = 0

data\_true\_manual\_testing[’class’] = 1

data\_fake\_manual\_testing.head(10)

data\_true\_manual\_testing.head(10)

data\_merge = pd.concat([data\_fake, data\_true], axis =

0)

data\_merge.head(10)

data\_merge.columns

data = data\_merge.drop([’title’,’subject’, ’date’],

axis = 1)

data.isnull().sum()

data=data.sample(frac=1)

data.head()

data.reset\_index(inplace=True)

data.drop([’index’],axis=1,inplace=True)

data.columns

data.head()

def wordopt(text):

text = text.lower()

text = re.sub(’\[.\*?\]’, ’’, text)

text = re.sub("\\W", " ", text)

text = re.sub(’https?://\St|www\.|S+’, ’’, text)

text = re.sub(’<."?>+’, ’’, text)

text = re.sub(’[%s]’ % re.escape(string.punctuation),

’ ’,text)

text = re.sub(’\n’, ’’, text)

text = re.sub(’\w\*\d\w\*’, ’’, text)

return text

data[’text’] = data[’text’].apply(wordopt)

x = data[’text’]

y = data[’class’]

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

, test\_size= 0.25)

from sklearn.feature\_extraction.text import

TfidfVectorizer

vectorization = TfidfVectorizer()

xv\_train = vectorization.fit\_transform(x\_train)

xv\_test = vectorization.transform(x\_test)

from sklearn.linear\_model import LogisticRegression

LR = LogisticRegression()

LR.fit(xv\_train, y\_train)

pred\_lr = LR.predict(xv\_test)

LR.score(xv\_test, y\_test)

print(classification\_report(y\_test, pred\_lr))

from sklearn.tree import DecisionTreeClassifier

DT = DecisionTreeClassifier()

DT.fit(xv\_train, y\_train)

pred\_dt = DT.predict(xv\_test)

DT.score(xv\_test, y\_test)

print(classification\_report(y\_test, pred\_dt))

from sklearn.ensemble import GradientBoostingClassifier

GB=GradientBoostingClassifier(random\_state=0)

GB.fit(xv\_train,y\_train)

predit\_gb=GB.predict(xv\_test)

GB.score(xv\_test,y\_test)

print(classification\_report(y\_test, predit\_gb))

from sklearn.ensemble import RandomForestClassifier

RF=RandomForestClassifier(random\_state=0)

RF.fit(xv\_train,y\_train)

pred\_rf=RF.predict(xv\_test)

RF.score(xv\_test,y\_test)

print(classification\_report(y\_test, pred\_rf))

def output\_lable(n):

if n == 0:

return "Fake News"

elif n == 1:

return "Not A Fake News"

def manual\_testing(news):

testing\_news= {"text": [news]}

new\_def\_test = pd.DataFrame(testing\_news)

new\_def\_test["text"] = new\_def\_test["text" ].apply(

wordopt)

new\_x\_test=new\_def\_test["text"]

new\_xv\_test = vectorization.transform(new\_x\_test)

pred\_LR=LR.predict(new\_xv\_test)

pred\_DT =DT.predict(new\_xv\_test)

predit\_GB=GB.predict(new\_xv\_test)

pred\_RF=RF.predict(new\_xv\_test)

return print("In\nLR Prediction: {} \nDT Prediction

:{} \nGB Prediction: {} \nRF Prediction: {}".format

(output\_lable(pred\_LR[0]),

news = str(input())

manual\_testing(news)