Dissertation on

A Machine Learning Approach for Lung Cancer Detection

Thesis submitted towards partial fulfillment of the requirements for the degree

Master of Technology in IT (Courseware Engineering)

Submitted by

Yousuf Sk

EXAMINATION ROLL NO.: M4CWE24008

UNIVERSITY REGISTRATION NO.: 163778 of 2022-23

Under the guidance of

Mr. Joydeep Mukherjee

School of Education Technology

Jadavpur University

Course affiliated to

Faculty of Engineering and Technology

Jadavpur University

Kolkata-700032

INDIA

2024

M. TECH.IT (COURSEWARE ENGINEERING) Course affiliated to
Faculty of Engineering and Technology
Jadavpur University
Kolkata – 700032

CERTIFICATE OF RECOMMENDATION

This is to certify that the thesis titled **"**A Machine Learning Approach for Lung Cancer Detection" is an authentic work carried out by Yousuf Sk under our supervision and guidance as part of the requirements for the degree of Master of Technology in IT (Courseware Engineering) in School of Education Technology, during the academic session 2023-2024.

SUPERVISOR
School of Education Technology
Jadavpur University
Kolkata-700 032

DIRECTOR
School of Education Technology
Jadavpur University
Kolkata-700 032

DEAN-FISLM
Jadavpur University
Kolkata-700032

M. TECH. IT (COURSEWARE ENGINEERING) Course affiliated to
Faculty of Engineering and Technology
Jadavpur University
Kolkata – 700032

CERTIFICATE OF APPROVAL **

This foregoing thesis is hereby approved as a credible study of an engineering subject carried out and presented in a manner satisfactory to warrant its acceptance as a prerequisite to the degree for which it has been submitted. It is understood that by this approval the undersigned do not endorse or approve any statement made opinion expressed, or conclusion drawn therein but approve the thesis only for the purpose for which it has been submitted.

Committee of Final Examination
for Evaluation of the Thesis

** Only in case the thesis is approved.

DECLARATION OF ORIGINALITY AND ACADEMIC ETHICS COMPLIANCE

I hereby affirm that this thesis comprises both a literature review and original research conducted by myself as part of my Master of Technology in IT (Courseware Engineering) studies. All information presented in this document has been gathered and reported in adherence to academic standards and ethical principles.

I further declare that, in accordance with these standards and principles, I have fully acknowledged and cited all sources and results that are not my own original work.

NAME : YOUSUF SK

EXAMINATION ROLL NUMBER: : M4CWE24008

THESIS TITLE: : A Machine Learning Approach for Lung Cancer Detection

DATE: SIGNATURE:

ACKNOWLEDGMENT

First and foremost, I would like to convey my deep gratitude to my supervisor, **Mr. Joydeep Mukherjee**, for his steadfast support throughout my thesis work and related research. His guidance and invaluable recommendations have been crucial during the research process and in the writing of this thesis. I am immensely thankful for the encouragement and assistance he provided throughout the entire research period.

Additionally, I also wish to express my sincere thanks to **Prof. Dr. Matangini Chattopadhya**y and **Dr. Saswati Mukherjee** for their consistent encouragement and backing during my time at the School of Education Technology. I am appreciative of the entire staff, including the Lab Assistants, and all those who contributed to my work through their cooperation and insights.

I am also deeply grateful to all my classmates in the M.Tech. IT (Courseware Engineering) program and the Master in Multimedia Development courses for their suggestions and ongoing support.

I extend my heartfelt thanks to my parents for their unwavering support through every challenge during my studies. Lastly, I am thankful to my friends and well-wishers who have always had faith in me and offered their support.

Yousuf Sk

Examination Roll No: **M4CWE24008**

Registration No. **163778** of **2022-2023**

M. Tech. IT (Courseware Engineering)

School Of Education Technology

Jadavpur University, Kolkata-700032

Table of Contents

Title Page

List of Diagrams

List of Table

EXECUTIVE SUMMARY

Early detection of lung cancer plays a pivotal role in improving patient outcomes and reducing mortality rates.

This research presents a novel approach utilizing machine learning techniques for the early detection of lung cancer based on symptom-based numerical attributes. Using a dataset containing numeric characteristics derived from indications such as chronic disease, fatigue, allergy, wheezing, coughing, shortness of breath, swallowing difficulty, and chest pain the proposed machine learning models aim to detect the presence of lung cancer in individual patients.

This comprehensive investigation validates the effectiveness of machine learning algorithms in the early detection of potential lung cancer cases. This research showcases the potential of merging machine learning with symptom-based datasets for lung cancer detection, thereby advancing prognosis and patient treatment. This research employs ANN, Logistic Regression, Gaussian Naive Bayes Classifier, and SVM conducting a thorough comparison to identify the optimal model for lung cancer detection. The significance of hyperparameter tuning in optimizing the accuracy and reliability of each model is highlighted. Trade-offs between different machine learning models, particularly in terms of sensitivity, specificity, and overall predictive power, are thoroughly examined. After evaluating the performance of each model the feature importance of each attribute is determined and shown with data visualization techniques.

CHAPTER 1

1. Introduction

1.1 Overview

The prevalence and lethality of lung cancer present a significant global health challenge, with the disease ranking among the most diagnosed and deadliest forms of cancer worldwide. According to the World Health Organization (WHO), an alarming 2.2 million new cases of lung cancer emerged globally in 2020 alone, underscoring the urgent need for effective detection and treatment strategies. Regrettably, lung cancer remains a leading cause of cancer-related deaths, claiming nearly 1.8 million lives globally in the same year, constituting approximately 18% of all cancer-related deaths. Early detection stands as a critical determinant in improving patient outcomes and mitigating mortality rates linked with lung cancer.
However, invasive procedures are often relied upon by conventional diagnostic methods, and the disease may not be detected until it has reached an advanced stage, limited treatment options and compromising survival rates. In recent years, the burgeoning interest in harnessing machine learning techniques for early lung cancer detection has been recognized as a promising avenue.

The capability of machine learning to sift through vast datasets, including symptom-based attributes, is leveraged to pinpoint individuals at risk of developing the disease. Numerical attributes extracted from symptoms such as yellow fingers, anxiety, chronic disease, fatigue, allergy, wheezing, coughing, shortness of breath, swallowing difficulty, and chest pain are utilized by machine learning models to discern intricate patterns and relationships within the data, facilitating accurate classification of individuals at risk of lung cancer. A novel approach to lung cancer detection is embarked upon in this paper through the lens of machine learning algorithms. A robust and precise model capable of identifying potential instances of lung cancer at an early stage is aimed to be crafted by utilizing a dataset replete with symptom-based numerical attributes. The performance of four distinct machine learning algorithms—Artificial Neural Network (ANN), Logistic Regression, Gaussian Naive Bayes Classifier, and Support Vector Machine (SVM)—will be rigorously scrutinized. The efficacy of these models will be juxtaposed to ascertain the most effective model for lung cancer detection. A systematic framework of experimentation and validation will be followed in this study to showcase the efficacy of machine learning in augmenting lung cancer detection, thereby fostering improved prognosis and patient care. The power of machine learning, in tandem with symptom-based datasets, will be harnessed with the aspiration of paving the way for more potent screening and early intervention strategies for lung cancer. Ultimately, this endeavour is propelled by the vision of saving lives and enhancing the quality of patient outcomes in the realm of lung cancer diagnosis and treatment.

1.2 Problem Statement

Lung cancer's high mortality rates and delayed diagnosis are called for improved detection methods. Machine learning is employed to analyze symptom-based data, to enhance early detection. The goal is to improve patient outcomes, lower mortality rates, and address lung cancer as a pressing public health issue. A method is also introduced to calculate individual lung cancer probability by examining patient-specific symptoms and risk factors.

1.3 Objective of the Research Work

- The objectives of this research work are as follows:
- A machine learning-based approach for early lung cancer detection is developed and evaluated.
- Numerical attributes associated with potential risk factors for lung cancer are utilized.
- Demographic information, medical history, lifestyle factors, and potential symptoms are comprised.
- The performance of four machine learning algorithms—ANN, Logistic Regression, Naïve Bayes Classifier, and SVM—is evaluated.
- Individuals at risk of lung cancer at an early stage are accurately identified.
- Rigorous experimentation and validation are conducted to assess model efficacy.
- Contributions are made to improve prognosis and patient care through enhanced lung cancer detection.

1.4 Assumptions and Scopes

1.4.1 Assumptions

The assumptions for this research could include:

- Validity of Symptom-based Numerical Attributes: It assumes that the symptom-based numerical attributes used for analysis accurately represent the presence and severity of symptoms associated with lung cancer.

- Representativeness of the Dataset: It assumes that the dataset used for training the machine learning model is comprehensive and representative of the diverse population affected by lung cancer.

- Generalizability of Findings: It assumes that the findings and conclusions drawn from the machine learning-based approach can be generalized to broader populations and healthcare settings.

- Quality of Data Collection: It assumes that the data collection process for obtaining symptom-based numerical attributes is reliable and free from biases or errors.

- Applicability of Machine Learning Algorithms: It assumes that the selected machine learning algorithms are suitable for analyzing the dataset and can effectively identify patterns indicative of early-stage lung cancer.

- Availability of Follow-up Data: It assumes access to follow-up data to assess the accuracy of the machine learning-based approach in predicting lung cancer outcomes and patient prognosis.

- Clinical Validation: It assumes that the machine learning-based approach will undergo rigorous clinical validation to assess its performance against existing diagnostic methods and its potential impact on patient outcomes.

1.4.2 Scopes

- Evaluation of Machine Learning Models: Assessing the effectiveness of Logistic Regression, ANN, SVM, and Naive Bays algorithms in detecting lung cancer using symptom-based numerical attributes.

- Impact on Early Intervention: Investigating the potential of machine learning to improve prognosis and patient care through enhanced screening and early detection strategies for lung cancer.

1.5 Concept and Problem Analysis

Using machine learning for lung cancer prediction based on symptom-based attributes offers significant advantages over traditional techniques for lung cancer detection such as Computed Tomography, PET Scan, etc. By analyzing datasets encompassing symptoms like yellow fingers, anxiety, wheezing, and more, machine learning models can discern intricate patterns and relationships, facilitating accurate risk classification. This approach enables early detection by identifying subtle indicators often missed by conventional methods, leading to timely intervention and improved patient outcomes. Moreover, machine learning allows for personalized risk assessment, considering individual characteristics and symptoms to tailor screening and prevention strategies. Automated analysis streamlines the screening process, reducing the need for invasive procedures and optimizing healthcare resource utilization.

So, a machine learning model has been proposed for lung cancer prediction based on symptom-based attributes which will not only predict if a patient has lung cancer or not but also it will calculate the percentage of chance of a patient having lung cancer.

The proposed machine learning model for lung cancer prediction based on symptom-based attributes has been proposed to revolutionize early detection and risk assessment. By harnessing advanced algorithms, the model not only determines the presence or absence of lung cancer but also quantifies the probability of its occurrence. It achieves this by unravelling intricate connections

and subtle patterns inherent in the dataset, discerning the nuanced relationships between various symptoms and their predictive value for lung cancer.

Through comprehensive analysis, the model navigates through vast amounts of data, identifying key indicators and their respective contributions to the prediction process. Its ability to recognize complex interdependencies enables a deeper understanding of the underlying factors influencing lung cancer risk. By quantifying the likelihood of lung cancer occurrence, the model provides clinicians and patients with valuable insights, guiding informed decision-making and facilitating proactive intervention strategies.

In essence, this model serves as a powerful tool in the fight against lung cancer, offering not only accurate predictions but also a nuanced understanding of the intricate web of symptoms associated with the disease. Its capabilities hold promise for enhancing early detection efforts, improving patient outcomes, and ultimately, saving lives.

The advantages of Lung Cancer Detection using clinical attributes or patients' medical history

The advantages of this research work:

1. Early Diagnosis: Identifying lung cancer early can significantly improve treatment outcomes and survival rates.

2. Non-Invasive Methods: Relying on symptoms and clinical attributes can offer a non-invasive and cost-effective alternative to more invasive procedures like biopsies or expensive imaging techniques.

3. Improving Screening Programs: Enhancing the effectiveness of lung cancer screening programs by incorporating symptom-based assessments can help reach a larger population.

4. Personalized Healthcare: Understanding lung cancer's clinical attributes and symptoms can lead to more personalized and targeted healthcare interventions.

5. Reducing Healthcare Costs: By facilitating early and accurate detection, this research can help reduce overall healthcare costs associated with late-stage cancer treatments.

1.6 Organization of Thesis

Chapter 1- This chapter includes an introduction of the thesis which consists of overview, problem statement, the objective of the research work, assumptions, and scopes.

Chapter 2- This chapter includes literature surveys done to complete the research work.

Chapter 3- This chapter includes the proposed approach that has been used to detect lung cancer in individual patients using clinical attributes.

Chapter 4 -This chapter includes the experimentations and results of the research.

Chapter 5- This chapter contains comparative analysis of all the proposed machine learning models.

Chapter 6- This chapter includes the conclusion and the future scope of the thesis.

References- All the references are listed here.

Appendix- All the codes are provided here.

CHAPTER 2

2. LITERATURE SURVEY

M. Nasser and S. S. Abu-Naser [1], proposed research that implements an Artificial Neural Network (ANN) to predict lung cancer based on various symptoms such as yellow fingers, anxiety, and chronic disease. Utilizing the "survey lung cancer" dataset, the ANN was trained and validated to assess its predictive capability. The model demonstrated high effectiveness, achieving an accuracy rate of 96.67% in detecting lung cancer, thereby affirming its reliability for medical diagnostic applications.

Sang Min Park [2], proposed research on pre-diagnosis factors like smoking, alcohol consumption, obesity, and insulin resistance highlighting their significant impact on cancer survival. A study involving over 14,000 male patients revealed that smoking and high fasting serum glucose levels are associated with poorer outcomes for various cancers, while a higher body mass index correlates with longer survival in specific cancers. These findings emphasize the critical role of lifestyle factors in cancer prognosis.

Xing, P-Y, Zhu [3], proposed a decade-long nationwide retrospective study in China that explored the relationship between symptoms and the diagnosis of lung cancer, focusing on non-small cell lung cancer (NSCLC). The research analysed data from lung cancer patients diagnosed between 2005 and 2014. Using multivariate unconditional logistic regression, the study calculated odds ratios to determine the association between symptoms, physical signs, and lung cancer diagnosis in the Chinese population.

F M Walter [4], proposed a prospective cohort study that analysed factors affecting the time to lung cancer diagnosis and its stage at detection. Among 963 patients, 15.9% were diagnosed with primary lung cancer. Haemoptysis, present in 21.6% of cases, was the most significant symptom linked to lung cancer. The study found that diagnostic intervals were shorter for cancer cases, especially those with late-stage disease, underscoring the need for early detection programs focusing on symptom progression.

V. Krishnaiah [5], proposed this study which explores classification-based data mining techniques such as Rule-based, Decision Tree, Naïve Bayes, and Artificial Neural Networks for diagnosing lung cancer. It emphasizes the use of advanced methods like ODANB and NCC2, which handle imprecise probabilities and small datasets. The research aims to leverage large volumes of healthcare data to uncover hidden patterns and improve the accuracy of early cancer detection, ultimately aiding in better diagnosis and patient outcomes.

S. P. Maurya [6], proposed research comparing twelve machine learning algorithms for predicting lung cancer using clinical data with eleven symptoms and two major patient habits. The study found that the K-Nearest Neighbor Model and Bernoulli Naive Bayes Model were the most effective for early detection. This approach aims to improve diagnosis accuracy in the context of rising PM 2.5 levels and the challenge of detecting lung cancer at an early, manageable stage.

T. Divya [7], proposed research introducing the Integrated Deep Learning-based Enhanced Grey Wolf Optimization (IDL-EGWO) for early lung cancer detection. This method addresses the instability of the Grey Wolf Optimizer by using a weighted average approach to improve convergence and prevent local optima issues. Coupled with an Artificial Neural Network, the IDL-EGWO achieved a notable accuracy of 97%, outperforming previous techniques in precision, recall, and overall performance.

Adrian Levitsky [8], proposed research to identify early predictive symptoms for primary lung cancer using an interactive e-questionnaire. Analysed data from 506 patients led to the selection of 63 key symptoms, such as variable cough and back pain, combined with seven background factors like smoking and age. The model achieved an area under the ROC curve of 0.767, demonstrating the potential for improving lung cancer prediction and diagnostic decision-making.

Jia Liao [9], proposed research to assess postoperative symptoms in lung cancer patients at discharge. Analyzing data from 366 patients, the study identified core symptoms such as cough, pain, disturbed sleep, shortness of breath, and fatigue, affecting over half of the participants. Factors like low income and the use of two chest tubes were linked to higher symptom severity. The study highlights the need for targeted care for patients with these conditions.

Rachel D. McCarty [10], proposed research to explore lung cancer diagnosis pathways among unscreened individuals. By interviewing 13 patients and 13 healthcare providers, the study identified barriers and facilitators across appraisal, help-seeking, and diagnostic stages. Barriers included symptom minimization and care hesitancy, while facilitators involved symptom acknowledgment and routine care. Challenges within the health system and social determinants also impacted diagnosis, highlighting the need for improved early detection strategies.

Nasareenbanu Devihosur [11], proposed research to enhance lung cancer diagnosis using machine learning algorithms. The study utilized Logistic Regression, Gradient Boost, LGBM, and Support Vector Machine to overcome limitations in current diagnostic methods. The approach aimed to improve diagnostic accuracy and efficiency, leading to earlier and more tailored treatments. The research demonstrated promising results, with Random Forest achieving 97% accuracy, highlighting its effectiveness in analysing complex lung cancer data for better patient outcomes.

Manju B R [12], proposed research to develop an interactive learning framework for disease detection using Principal Component Analysis (PCA) and Support Vector Machines (SVM). PCA effectively classifies target classes by reducing dimensionality, while SVM excels in classification with limited data. The study combines these techniques to enhance early detection of malignancies, with performance evaluated through a confusion matrix. The developed model demonstrated high accuracy and robustness in identifying different stages of disease.

Trailokya Raj Ojha [13], proposed research to address the challenge of lung cancer forecasting using various machine learning algorithms. The study applied Support Vector Machine, Adaptive Boosting, k-Nearest Neighbor, Logistic Regression, J48, and Naïve Bayes to medical history and physical activity data. The results showed that all algorithms effectively predicted lung cancer rates, with Logistic Regression achieving the highest performance, boasting an accuracy and f-measure of 94.7%.

Swati Mukherje [14], introduced a study aimed at improving lung disease diagnosis through advanced methods. The research focuses on utilizing a neural network model for identifying cancerous cells in medical images, a common challenge in therapeutic imaging. By developing a lung cancer detection system based on AI and deep learning, particularly with CNN classification, the study enhances precision in diagnosis. The system involves image acquisition, preprocessing, enhancement, segmentation, feature extraction, and neural network identification, offering a cost-effective solution for better decision support in lung cancer treatment.

Elias Dritsas [15], put forward a study focusing on lung health, highlighting the crucial role of the lungs in oxygen distribution and filtration. Despite natural defenses, the lungs remain vulnerable to diseases like cancer. This research employed machine learning techniques to develop effective models for identifying individuals at high risk of lung cancer, enabling earlier interventions. The proposed Rotation Forest model demonstrated exceptional performance, achieving an AUC of 99.3% and an F-Measure, precision, recall, and accuracy of 97.1%.

Atharva Bankar [16], proposed a study examining the application of machine learning in healthcare, particularly for early cancer detection. Various algorithms, including KNN, SVM, Decision Trees, and Random Forest, were utilized to assess cancer presence based on patient symptoms. The research analysed symptoms across different age groups—Youth, Working Class, and Elderly—using tree-based algorithms like Decision Trees, Random Forest, and XGBoost. The study found that factors such as coughing of blood, clubbing of fingernails, genetic risk, passive smoking, and snoring are commonly associated with lung cancer across all age groups.

Baidaa Al-Bander [17], proposed a study focusing on enhancing lung cancer risk prediction using a web-based survey. The research developed a multi-criteria decision support system combining the Analytical Hierarchy Process (AHP) with an Artificial Neural Network (ANN). The AHP assigns weights to individual cancer symptoms based on survey data, which are then used to train a multi-layer perceptron ANN. The evaluation on 276 subjects demonstrated promising results in prediction accuracy across various metrics.

Sandro J Martins [18], put forward a study examining the use of the Lung Cancer Symptom Scale (LCSS) and pulse oximetry to evaluate lung cancer patients undergoing standard anticancer therapy. Involving forty-one patients, the research developed a survival model using variables such as SpO2 levels, LCSS scores, and Karnofsky performance status. Results showed that SpO2 > 90% and LCSS appetite and fatigue scores were independent survival predictors. Further investigation of these factors as prognostic indicators is recommended.

Radhanath Patra [19], put forward a study addressing the challenge of lung cancer, which is increasing in both youth and elderly despite advanced medical facilities. The study highlights the need for early detection and diagnosis. It analyses various machine learning classifiers on UCI lung cancer data, comparing their effectiveness. The RBF classifier demonstrated high accuracy of 81.25%, proving to be an effective tool for predicting lung cancer.

Yasemin Gültepe [20], conducted a study to enhance lung cancer diagnosis using machine learning. The research, involving 32 × 56 sized data from the UCI Machine Learning Repository, focused on improving classification accuracy through effective pre-processing methods. Among nine datasets and six classifiers, the k-nearest neighbors algorithm outperformed others like random forest and naive Bayes. Notably, Z-score normalization (83% accuracy), principal component analysis (87%), and information gain (71%) were the most effective pre-processing methods.

CHAPTER 3

3. Proposed Approach

For this research work various supervised machine learning models have been used ANN, Logistic Regression, Naïve Bayes Classifier. For this study, at first the records of 309 patients having 16 clinical attributes are taken from Kaggle which are also used in the base paper of this research. At first, to compare the performance of the ANN model in order to predict lung cancer used in the base paper the same data is used in the ANN model. Then another dataset taken from the data world is concatenated with the previous dataset after some data preprocessing steps are performed on it. Then some machine learning algorithms as- ANN, Logistic Regression, Naïve Bayes Classifier and SVM are defined, trained and tested with the new concatenated dataset to predict lung cancer in the individual patient. Then each model is used to calculate the probability of having lung cancer for each patient. The Confusion Matrix that was used by the models has calculated four different tables - true positive (TP), true negative (TN), false positive (FP), and False Negative (FN).

3.1 Dataset Description

For this study, two datasets consisting of patient's clinical attributes including symptoms related to cancer are collected from Kaggle and data world. Then based on the matching column these datasets are concatenated. The independent attributes in the final concatenated dataset are age, smoking, chronic disease, fatigue, allergy, wheezing, alcohol consuming, coughing, shortness of breath, swallowing difficulty, chest pain and one target variable., lung cancer. These symptoms are diagnosed by the model to detect lung cancer in patients.

For this research, two comprehensive datasets from Kaggle and Data World consisting of clinical attributes and symptoms related to lung cancer are collected. The total no of data rows in the final concatenated dataset is 1000 and no of columns are 25 from which 24 are independent features or risk factor and one is target variable. The dataset includes a variety of attributes that are pivotal in understanding and diagnosing lung cancer. After preprocessing the attributes are:

age: Age of the patients, as the risk of lung cancer increases with age.

smoking: Smoking status, a significant risk factor for lung cancer.

chronic disease: Presence of chronic diseases, which can complicate or contribute to lung cancer development.

fatigue: Chronic fatigue, a common symptom in various cancers, including lung cancer.

allergy: History of allergies, which might impact respiratory health.

wheezing: Wheezing sounds in the lungs, indicative of respiratory distress.

alcohol consuming: Alcohol consumption habits, as excessive alcohol intake can impact overall health and cancer risk.

coughing: Persistent coughing, a primary symptom of lung cancer.

shortness of breath: Difficulty breathing, is another significant symptom of lung cancer.

swallowing difficulty: Difficulty in swallowing, which can be a sign of advanced lung cancer.

chest pain: Persistent chest pain, often associated with lung cancer.

Using this dataset, the model aims to diagnose lung cancer by analyzing these clinical attributes and symptoms. The diversity and relevance of these attributes allow the model to identify patterns and correlations that may not be immediately evident through traditional diagnostic methods. By leveraging machine learning techniques, the model can process this complex data and provide a more accurate and timely diagnosis of lung cancer.

Dataset Description:

Table 1: Attribute Description

Feature	Levels
gender	M=male, F=female
age	Age of the patient
smoking	yes=2, no=1
chronic disease	yes=2, no=1
fatigue	yes=2, no=1
allergy	yes=2, no=1
wheezing	yes=2, no=1
alcohol consuming	yes=2, no=1
coughing	yes=2, no=1
shortness of breath	yes=2, no=1
swallowing difficulty	yes=2, no=1
chest pain	yes=2, no=1
lung cancer	yes=2, no=1

	GENDER	AGE	SMOKING	YELLOW_FINGERS	ANXIETY	PEER_PRESSURE	CHRONIC DISEASE	FATIGUE	ALLERGY	WHEEZING	ALCOHOL CONSUMING	COUGHING	SHORTNESS OF BREATH	SWALLOWING DIFFICULTY	CHEST PAIN	LUNG_CANCER
0	M	69	1	2	2	1	1	2	1	2	2	2	2	2	2	YES
1	M	74	2	1	1	1	2	2	2	1	1	1	2	2	2	YES
2	F	59	1	1	1	2	1	2	1	2	1	2	2	1	2	NO
3	M	63	2	2	2	1	1	1	1	1	2	1	1	2	2	NO
4	F	63	1	2	1	1	1	1	1	2	1	2	2	1	1	NO
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
297	F	47	2	2	1	2	2	2	2	2	1	2	2	1	1	YES
298	M	62	2	1	2	1	1	2	1	2	2	2	2	1	2	YES
299	M	65	2	2	2	2	1	2	2	1	1	1	2	2	1	YES
300	F	63	2	2	2	2	2	2	2	2	1	2	2	2	2	YES
301	M	64	1	2	2	2	1	1	2	1	2	1	1	2	2	YES

Fig1: 1st (Kaggle) Dataset Visualization

	Patient Id	Age	Gender	Air Pollution	Alcohol use	Allergy	chronic Pain	Balanced Diet	Obesity	Smoking	Chest Pain	Fatigue	Weight Loss	Shortness of Breath	Wheezing	Swallowing Difficulty	Coughing	Lung Cancer
0	P1	33	1	1	1	2	1	1	1	1	1	1	1	1	1	1	1	No
1	P100	35	1	1	2	2	1	2	2	1	1	2	2	2	1	1	2	Yes
2	P1000	37	1	2	2	2	2	2	2	2	2	1	1	1	1	1	2	Yes
3	P101	46	1	2	2	2	2	2	2	2	2	1	1	1	1	1	1	Yes
4	P102	35	1	1	2	2	1	2	2	1	1	2	2	2	1	1	2	Yes
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
653	P995	44	1	2	2	2	2	2	2	2	2	2	1	1	2	2	2	Yes
654	P996	37	2	2	2	2	2	2	2	2	2	2	2	2	2	1	1	Yes
655	P997	25	2	1	2	2	1	2	2	1	1	2	2	2	1	1	2	Yes
656	P998	18	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	Yes
657	P999	47	1	2	2	2	1	2	2	1	1	2	2	2	1	1	2	Yes

Fig2: Visualization of 2nd Pre-processed Dataset (from Data World)

	age	smoking	chronic disease	fatigue	allergy	wheezing	alcohol consuming	coughing	shortness of breath	swallowing difficulty	chest pain	lung_cancer
0	69	1	1	2	1	2	2	2	2	2	2	YES
1	74	2	2	2	2	1	1	1	2	2	2	YES
2	59	1	1	2	1	2	1	2	2	1	2	NO
3	63	2	1	1	1	1	2	1	1	2	2	NO
4	63	1	1	1	1	2	1	2	2	1	1	NO
...	...	...	...	...	...	...	...	...	...	...	...	...
955	44	2	2	2	2	2	2	2	1	2	2	Yes
956	37	2	2	2	2	2	2	1	2	1	2	Yes
957	25	1	1	2	2	1	2	2	2	1	1	Yes
958	18	2	2	1	2	1	2	1	1	1	2	Yes
959	47	1	1	2	2	1	2	2	2	1	1	Yes

960 rows × 12 columns

Fig 3: Concatenated Dataset Description

3.2 Data Preprocessing for Kaggle Dataset

Data preprocessing is the initial step in the data analysis pipeline where raw data is transformed, cleaned, and organized to make it suitable for further analysis.
In this research, the method of data preprocessing is used twice once while evaluating the performance of the ANN model on the 1st dataset (from Kaggle) and another time on the final concatenated dataset after merging the 1st (from Kaggle) and 2nd dataset (from Data World).

So, here first the process is discussed for 1st dataset then the concatenated dataset.
This process involves tasks, they are as follows:

3.2.1 Exploratory Data Analysis

Exploratory Data analysis is the first step of data pre-processing. Exploratory Data Analysis (EDA) is a technique used to examine data sets, summarizing their key features and frequently using visual tools to do so. The primary goal of EDA is to understand the data and its underlying structure, patterns, and relationships, without making assumptions about the data distribution or relationships between variables.
In this stage first, the dataset is imported to the Jupyter Notebook.

Then, mean, minimum, maximum, standard deviation of every attribute are calculated i.e., needed to detect the outliers of every attribute.

3.2.2 Data Cleaning

Data cleaning, also known as data cleansing, is the process of identifying, correcting, and removing errors, inconsistencies, and inaccuracies from raw data to improve its quality, reliability, and usability for analysis or modelling purposes. The goal of data cleaning is to ensure that the dataset is accurate, complete, and consistent, thereby minimizing the risk of misleading results and erroneous conclusions in subsequent data analysis.

For this research work the steps that have been implemented to clean the dataset are as follows:

- **Handling Missing Values:**

A missing value, also known as a null value, is a placeholder used to indicate that no data is present for a particular variable or observation within a dataset. Missing values commonly occur due to various reasons such as data entry errors, equipment malfunctions, survey non-responses, or simply because the information was not collected.

The dataset is checked using certain commands for null or missing values but there are no such values present in the dataset.

- **Outliers Detection:**

Outliers are data points that differ markedly from the majority of observations within a dataset. These data points are notably different from the majority of the data and can potentially distort statistical analyses, machine learning models, or data visualization interpretations. Outliers can occur due to various reasons, including measurement errors, experimental variability, or genuine but rare events in the data.

The dataset is checked with certain commands to detect any outliers. The method that is used to check the outliers is Z-Score method.

- **Z-Score Method:**

This method detects outliers using the Z-Score method.

Here's an explanation of each step of how it works for the case of the column 'AGE' of the 1st dataset.

- **Calculation of Mean and Standard Deviation:**

The mean (`mean`) and standard deviation (`std`) of the data are pre-calculated. In this case, the mean value of the attribute 'AGE' is `62.705298`.

- **Loop Through Each Data Point:**

The method loops through each data point (`i`) in the input data (`AGE`).

- **Calculation of Z-Score:**

For each data point, the Z-Score is calculated using the formula:

$$Z=(X-\mu)/\sigma \text{----------(1)}$$

where `i` is the data point, `mean` is the mean of the data, and `std` is the standard deviation of the data.
The z-score measures how many standard deviations a data point is in terms of standard deviations.

- **Comparison with Threshold:**

If the absolute value of the Z-Score (`np. abs(score)`) is greater than a specified threshold (in this case, `3`), the data point is considered an outlier.
A z-score greater than 3 (or less than -3) is often used as a threshold to identify extreme outliers, as it corresponds to approximately 99.7% of the data falling within 3 standard deviations of the mean in a normally distributed dataset.

- **Appending Outliers:**

If a data point's Z-Score exceeds the threshold, it is appended to the `outliers` list.

- **Returning Outliers List:**

Finally, the method returns the list of outliers found in the input data.
In this case for the 'AGE' attribute the outlier is 21'.
The same process is used for each column of the attributes to detect the outliers.
This method effectively identifies outliers by quantifying how far each data point is from the mean in terms of standard deviations. Points that fall significantly beyond a specified threshold are considered outliers.

- **Dealing with the Outliers**

After detecting the outlier in the dataset, the record or row consisting of the outlier (For the attribute 'AGE'=21) is completely deleted. This will be repeated for each column.

The same process is followed in the case of concatenated dataset.

➢ **Shuffle the Dataset**

To ensure that the order of instances does not bias the learning algorithm during training the dataset is then shuffled using a certain algorithm.

3.3 Feature Extraction

Feature extraction is a key process in machine learning and data analysis that involves transforming raw data into a set of measurable and informative features or attributes. These features can then be used as input for machine learning algorithms to improve model performance and predictive accuracy. The main goal of feature extraction is to reduce the complexity of the data while retaining the most relevant information that represents the underlying patterns or structures.

The step where the dataset is split into features (**X**) and the target variable (**Y**) is crucial for preparing the data for machine learning. Here's a detailed explanation of why this step is needed:

Purpose of Splitting the Dataset into Features and Target Variable

➢ **Model Training and Prediction**:

- Features (X): These are the independent variables or predictors used to make predictions. In the context of lung cancer prediction, features might include various patient symptoms, medical history, and demographic information.

- Target Variable (Y): This is the dependent variable or the outcome that the model aims to predict. In this case, it indicates whether a patient has lung cancer or not.

➢ **Separation of Input and Output**:

- Machine learning models learn patterns and relationships between the input data (features) and the output data (target). By clearly separating **X** and **Y**, what the model should use as input and what it should predict is defined.
- This separation is essential for supervised learning, where the model is trained on labeled data (i.e., data with known outcomes).
- Splitting the dataset into **X** and **Y** allows for independent preprocessing of features and the target variable. For example, feature scaling, encoding categorical variables, and handling missing values can be done on **X**, while label encoding might be performed on **Y**.
- Here X refers all the clinical attributes and Y refers the target column 'LUNG_CANCER'.
- Ensures that the target variable does not get altered during preprocessing steps meant for features.

3.4 Data Splitting

Dividing data is an essential step in developing and assessing machine learning models. It involves dividing the available dataset into distinct subsets, typically known as the training set, and test set.

This process is crucial for:

➢ **Avoiding Overfitting**: Helps ensure the model learns general patterns, not just the noise in the training data.
➢ **Model Generalization**: Assesses how well the model performs on unseen data, reflecting its real-world applicability.
➢ **Unbiased Evaluation**: The test set provides an independent measure of model performance.

Here two machine learning splitting techniques are used in this research, they are:

Holdout Method: The holdout method is a straightforward technique for evaluating the performance of a machine learning model. It involves splitting the dataset into two or three distinct subsets: the training set, validation set, and test set.

3.4.1 Steps in the Holdout Method

➢ **Split the Dataset:**

- **Training Set:** Typically, 80% of the dataset. This subset is used to train the model, allowing it to learn the underlying patterns and relationships in the data.

- **Test Set:** Usually, 20% of the data. This subset is used only after the model has been trained and tuned. It provides an unbiased evaluation of the model's performance on new, unseen data, ensuring that the model generalizes well to real-world scenarios.

3.5 Encoding

Encoding is necessary in machine learning to convert categorical data into a numerical format that algorithms can understand and process. Most machine learning algorithms require numerical input to perform calculations and make predictions, so encoding ensures that categorical variables are represented in a way that the algorithms can use effectively.

For this research two types of encoding have been used, they are:

- **One Hot Encoding:**

One-hot encoding transforms categorical data into a binary (0 or 1) vector representation. Each category in a categorical feature is represented by a binary vector where only one bit is set to 1, indicating the presence of that category, and all other bits are set to 0.

Example: The lung cancer dataset consists of one independent attribute having categorical outcomes ‘Gender’ having two categorical classes ‘Male’ and ‘Female’. So, the classes of the attribute should be encoded to binary (‘0’and ‘1’). While one hot encoding the ‘Gender’ column is divided into two separate columns, ‘Gender_M’ and ‘Gender_F’. Now for an individual patient if ‘Gender_M’ is ‘0’ means value of ‘Gender_F’ will be ‘1’ refers that the gender of the patient is female.

- **Label Encoding:**

Label encoding is the process of converting categorical labels into numerical values. Each unique category is assigned an integer, which allows machine learning algorithms to process categorical data.

Label encoding is used for the target variable "lung cancer" in the lung cancer dataset to convert the categorical labels (e.g., "YES" and "NO") into numerical values (e.g., 1 and 0). This transformation is necessary because most machine learning algorithms require the target variable to be in a numeric format to perform classification tasks. Label encoding simplifies the representation and ensures compatibility with the algorithms.

- **Standardization:**

Standardization is the process of transforming data to have a mean of 0 and a standard deviation of 1. This ensures that all features contribute equally to the model by bringing them onto a common scale, making the data more suitable for many machine learning algorithms.

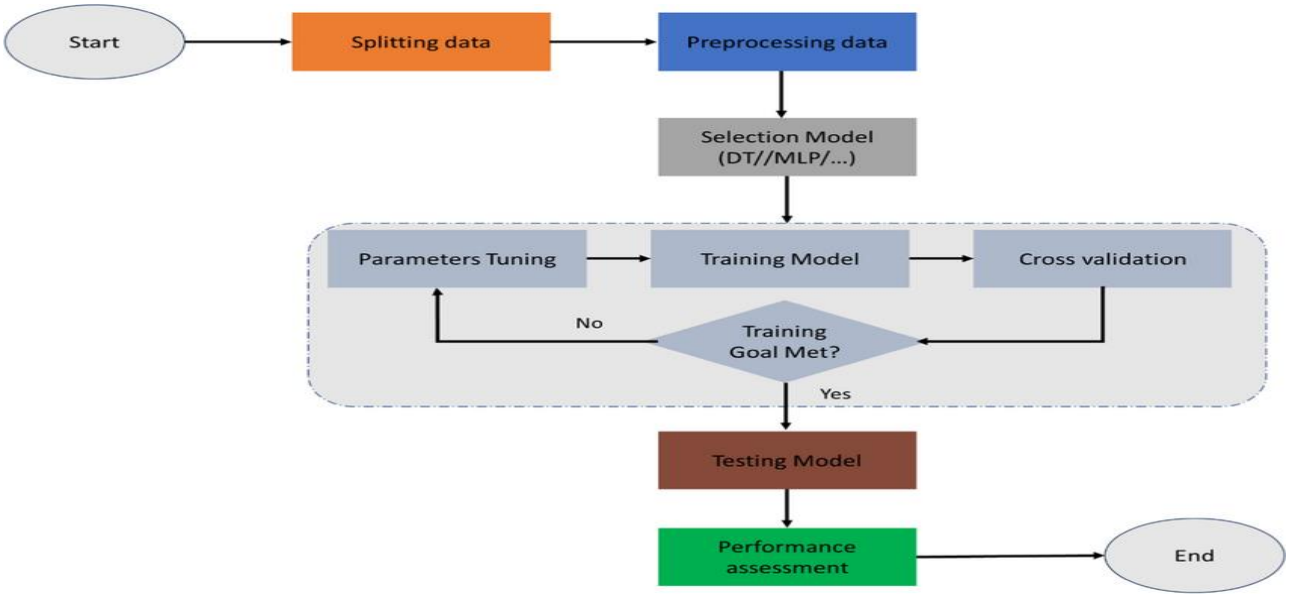

Fig4: Flow Chart of a General Machine Learning Model

Source:https://www.researchgate.net/figure/The-flow-chart-of-generalmachine learning-modeling_fig1_374291232

3.5.1 ANN for Lung Cancer Detection using Kaggle Dataset

An Artificial Neural Network (ANN) is a computational model influenced by the way biological neural networks in the human brain handle information. ANNs consist of interconnected layers of nodes (neurons) that work together to solve complex problems. They are particularly powerful in recognizing patterns and making predictions from data.

3.5.1.1 Structure of an ANN

- **Input Layer**:

The input layer consists of neurons that receive the features from the lung cancer dataset. Each neuron in this layer corresponds to one feature in the input data.

- **Hidden Layers**:

- These layers are positioned between the input and output layers. They consist of neurons that perform computations on the inputs. An ANN has multiple hidden layers, and the number of neurons in each layer can vary.

- Each neuron in a hidden layer takes inputs from all neurons in the previous layer, applies a weighted sum, adds a bias, and then applies an activation function (e.g., ReLU) to introduce non-linearity.

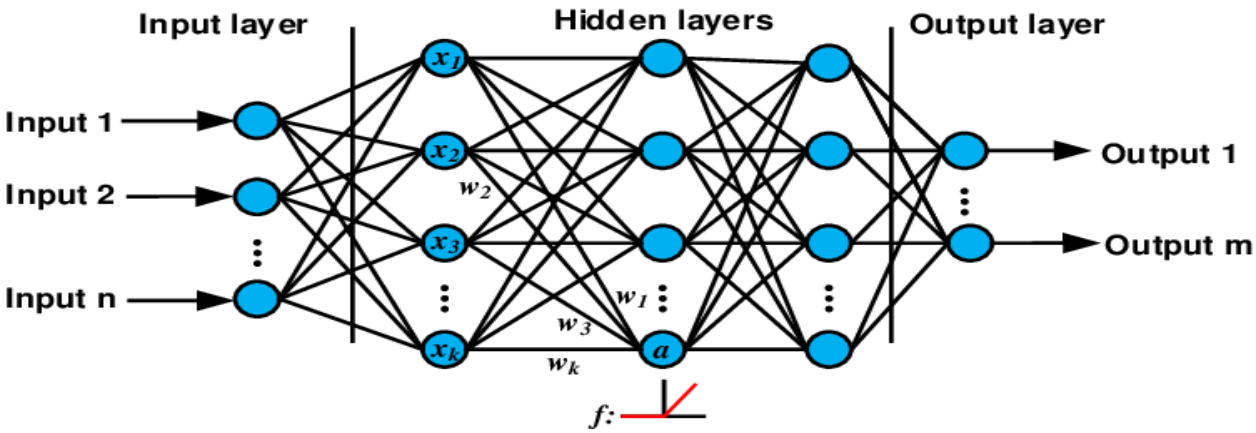

Fig 5: Structure of an ANN model

Source: https://www.researchgate.net/

- **Output Layer**:

The output layer produces the final prediction whether the patients consist of lung cancer or not. In the case of binary classification (like lung cancer detection), it typically has one neuron with a sigmoid activation function, which outputs a probability value indicating the likelihood of the positive class (e.g., the presence of lung cancer).

- **Mathematical Formulation:**

Cost Function:

In the context of artificial neural networks (ANNs), a cost function (also known as a loss function) is a mathematical function that measures the difference between the predicted output of the network and the actual target values. It quantifies how well the model's predictions match the true data, guiding the training process to minimize this difference and improve the model's accuracy.

For a classification problem, the cost function often used is the cross-entropy loss. The formula for the cross-entropy loss for a dataset with N training examples and C classes is:

$$J(\theta) = -\frac{1}{N} \sum_{j=1}^{N} \sum_{i=1}^{C} y_{ij} * \log(\hat{y}_{ij}) \text{----------(2)}$$

where:

- θ are the parameters of the model (weights and biases).
- N is the number of training examples.
- C is the number of classes.
- y_{ij} is a binary indicator (0 or 1) if class label i is the correct classification for the j-th training example.
- $\hat{y}_{ij}$ is the predicted probability that the j-th training example belongs to class i.

Now, in this research, based on the prediction of lung cancer the number of classes of the target variable is two ('Yes' and 'No').
So, for this ANN model used to detect lung cancer the modified formula of cost function is:

$$J(\theta) = -\frac{1}{N} \sum_{j=1}^{N} [y_j * \log(\hat{y}_j) + (1-y_j) * \log(1-\hat{y}_j)] \text{----------(3)}$$

where:

- y_j is the true label (0 or 1) for the j^{th} training example.
- $\hat{y}_j$ is the predicted probability that the j^{th} training example belongs to the positive class.
- $y_j * \log(\hat{y}_j)$, here if the true label y_j is 1, this term contributes to the loss. If $y_j=1$ and $\hat{y}_j$ is close to 0, this term will be large, indicating a poor prediction.
- $(1-y_j) * \log(1-\hat{y}_j)$, here if the true label y_j is 0, this term contributes to the loss. If $y_j=0$ and $\hat{y}_j$ is close to 1, this term will be large, indicating a poor prediction.

3.5.1.2 Working Principal ANN to Predict Lung Cancer

- **Forward Propagation**:

Data flows forward through the network, layer by layer. Each neuron computes a weighted sum of its inputs, adds a bias, and applies an activation function. This process transforms the input data as it passes through the hidden layers, enabling the network to learn complex patterns in the lung cancer dataset.

- **Activation Functions**:

  Activation functions like ReLU (Rectified Linear Unit) are applied in hidden layers to introduce non-linearity, allowing the network to model complex relationships between the input features and the target variable, i.e., ‘Lung Cancer’.

- **Backpropagation and Weight Updates**:

  The network uses backpropagation to adjust the weights and biases. During backpropagation, the gradients of the loss function with respect to each weight are calculated and used to update the weights in the direction that minimizes the loss. An optimizer (e.g., Adam) is used to perform the weight updates.

- **Training**:

  The network is trained over multiple epochs, where each epoch consists of forward and backward passes through the entire training dataset, i.e., the 80% of the total pre-processed dataset. The goal is to minimize the loss and improve the model's accuracy.

- **Testing**:

  After training, the model's performance is evaluated on the rest of the 20% pre-processed unlabelled test dataset to ensure it generalizes well to unseen data. Metrics like accuracy, precision and recall can be used to assess performance. The testing accuracy is 96.72%.

- **Prediction**:

  The final layer produces an output using a sigmoid activation function, which converts the weighted sum of inputs into a probability score between 0 and 1. This score indicates the likelihood of the presence of lung cancer. Typically, a threshold of 0.5 is used to assign class labels. If the probability output by the sigmoid function is greater than or equal to 0.5, the model predicts the positive class (e.g., "YES"). Otherwise, it predicts the negative class (e.g., "NO").

- **Feature Importance:**

  After assessing the model's performance on the combined dataset, key features of the model were identified. By employing specific code and data visualization techniques, the coefficient values for each attribute were calculated. A higher coefficient value indicates that the corresponding attribute is strongly associated with the target outcome, which in this case is lung cancer.

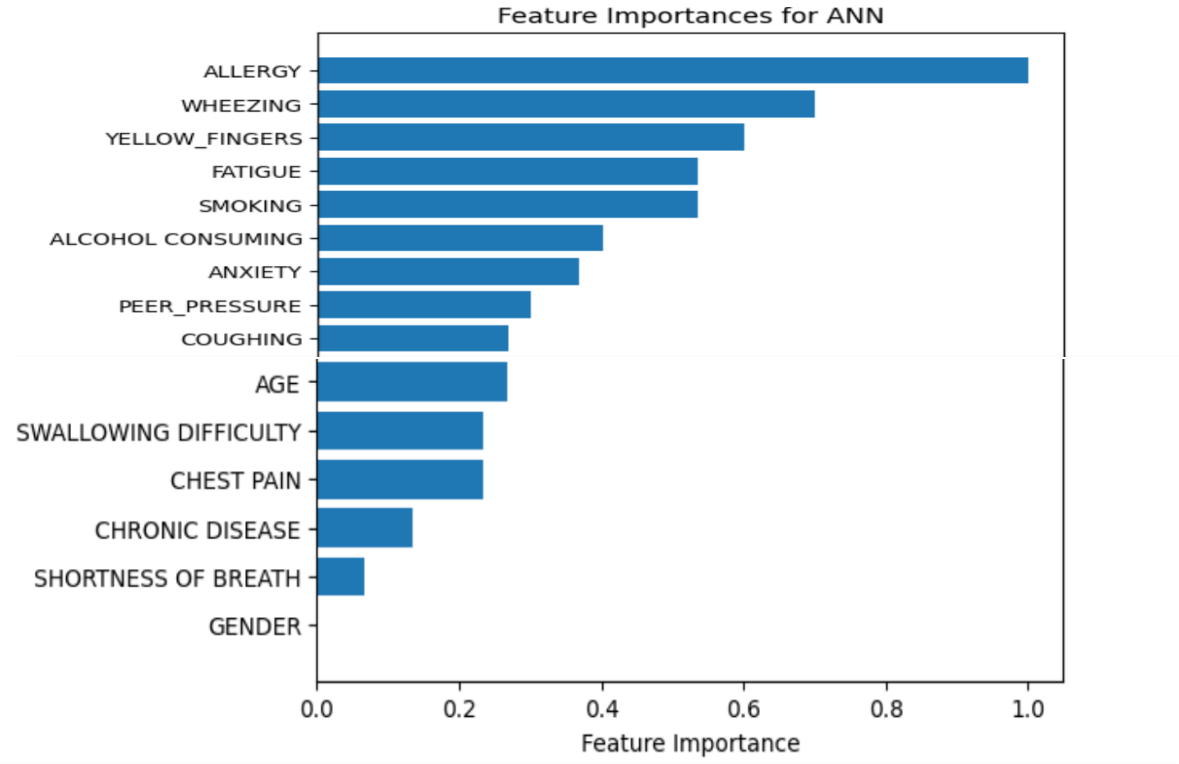

Fig6: Feature Importance of ANN

3.6 Machine Learning Models for Lung Cancer Detection Using Concatenated Dataset

Previously lung cancer prediction has been done with ANN on the Kaggle dataset.
Now, various machine learning algorithms like Logistic Regression, Naïve Bayes Classifier, and SVM including the ANN are used for lung cancer prediction on the concatenated dataset that is formed after merging the previously used Kaggle dataset to the dataset collected from data world. The dataset has 1000 rows and 25 columns.

3.6.1 Data Pre-processing

In this stage dataset is transformed into a usable format for training and testing a machine-learning model. The steps are as follows:

- **Data Concatenation:**

The two different datasets collected from Kaggle, and Data World are at first imported to the Jupytet Notebook environment. After that concatenation is performed with respect to the common columns present in both the dataset. The extra columns present in both datasets are to be discarded.

For the same, the column name and order of the columns of both datasets are transformed as same.

Then type conversion method is done to make the data type of both datasets same which is needed for merging them.
Now, implementing some certain algorithm the datasets are merged, and the concatenated dataset is formed.

- **Null Value Checking:**

In this process the concatenated dataset is checked if there is any missing value present in the dataset. But after implementing the certain algorithm it is noticed that there is no null value present in the dataset.

- **Dealing the Outliers:**

Using the previously mentioned Z-score method the dataset is checked if there are any outliers present in the concatenated dataset. But after implementing the certain algorithm it is noticed that there is no null value present in the dataset.

- **Shuffling:**

  Shuffling is performed on the dataset to ensure data randomness, preventing the model from learning spurious patterns due to data order. It helps create balanced training and validation sets, enhancing model generalization and reducing bias.

3.6.2 Feature Extraction

In this step the most important feature is retained to increase the model's performance and accuracy.

- **Separation of Independent and Dependent Attribute:**

  To provide the machine learning model, the exact independent features and the target variable to be trained and tested with accurately all the independent features (For example: age, smoking) are stored in X and the target variable (lung cancer) is stored in Y.

- **Splitting:**

  Using hold out method the entire pre-processed dataset is divided into training set and test set. In which the training set is given with the 80% of the total prep-processed concatenated data and the test set is given with the rest 20% data.

- **Encoding:**

  Encoding transforms categorical data into numerical format, enabling machine learning models to process and analyze the data effectively.

- **One Hot Encoding:**

  In the concatenated dataset there is no attribute with categorical value so there is no need to use one hot encoding.

- **Label Encoder:**

  The categorical target variable (lung cancer) is having two type of class -yes, and no. To convert these classes into numerical value label encoder is used.
  The class 'no' is converted to 0 and 'yes' is converted to 1.

- **Standardization:**

  Standard scaler is used on the independent feature dataset to standardize the features so that they can contribute equally. Standardization converts data so that it has a mean of 0 and a standard deviation of 1, ensuring all features are on the same scale.

➢ **Correlation:**

To find multicollinearity or dependency of an independent attribute on another independent attribute correlation-based feature selection method is used.
In the correlation matrix value in each cell represents the coefficient value of the correlation between two independent attributes. If the value of the coefficient between two independent attributes is greater than or equal to 0.9 then it is suggested that those attributes are highly correlated. And the attributes can be combined to form a new feature.
But in the pre-processed concatenated data there are no such attributes present.

After the steps involved in data preprocessing and feature extraction the pre-processed dataset is ready to use in the machine learning model.

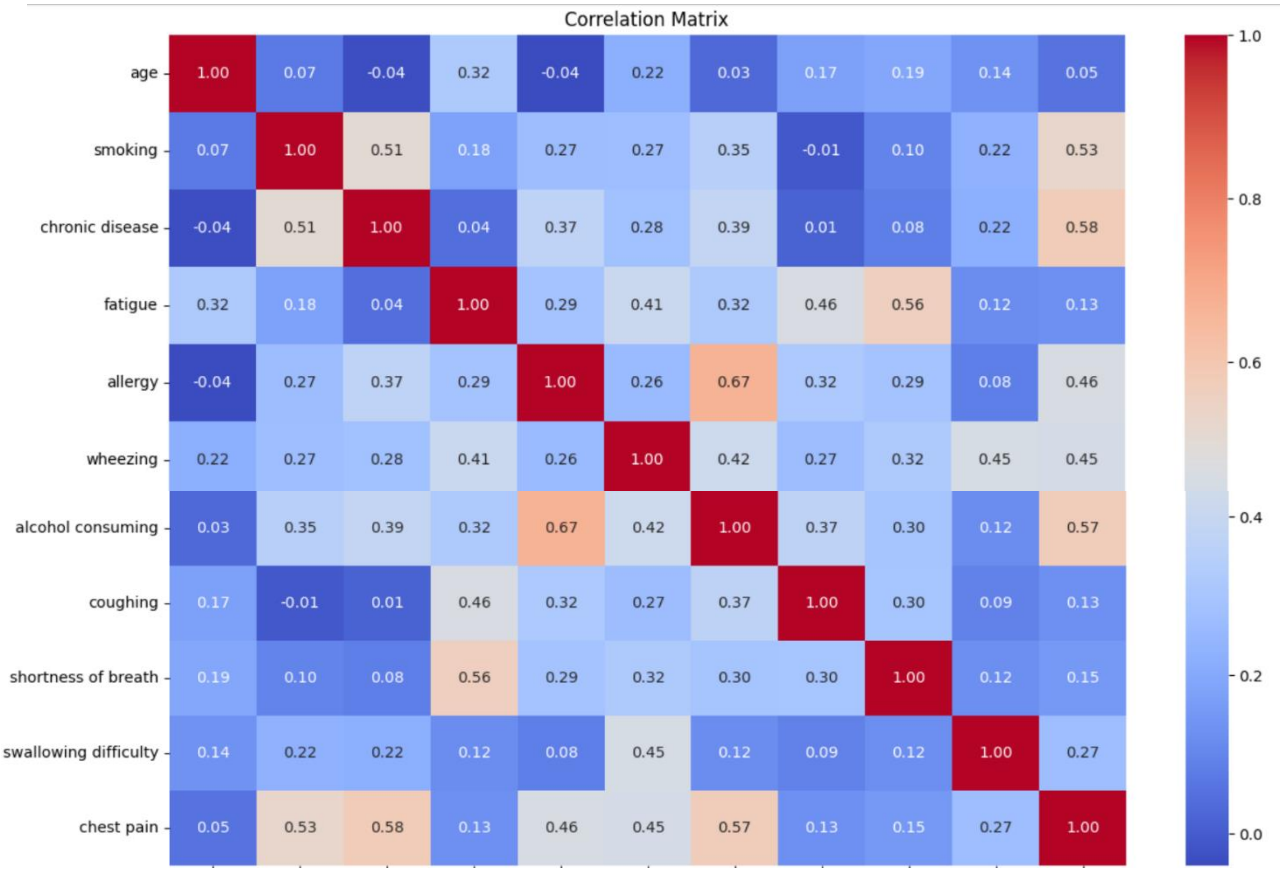

Fig 7: Correlation Matrix

3.6.3 Artificial Neural Network

Artificial neural network is used to make lung cancer classification for each individual patient after being trained and tested with the pre-processed concatenated Dataset.

- **Training**:

The network is trained over multiple epochs, where each epoch consists of forward and backward passes through the entire training dataset, i.e., the 80% of the total pre-processed concatenated dataset. The goal is to minimize the loss and improve the model's accuracy.

- **Testing**:

After training, the model's performance is evaluated on the rest of the 20% pre-processed unlabelled concatenated test dataset to ensure it generalizes well to unseen data. Metrics like accuracy, precision, and recall can be used to assess performance.

- **Prediction**:

The final layer produces an output using a sigmoid activation function, which converts the weighted sum of inputs into a probability score between 0 and 1. This score indicates the likelihood of the presence of lung cancer. Typically, a threshold of 0.5 is used to assign class labels. If the probability output by the sigmoid function is greater than or equal to 0.5, the model predicts the positive class (e.g., "yes"). Otherwise, it predicts the negative class (e.g., "no").

- **Feature Importance:**

After evaluating the model's performance on the concatenated dataset important features of the model are extracted. Using certain code and data visualization techniques the value of the coefficient of each attribute is measured. Here, the high value of the coefficient refers to the corresponding attribute highly related to the target outcome, i.e., lung cancer.

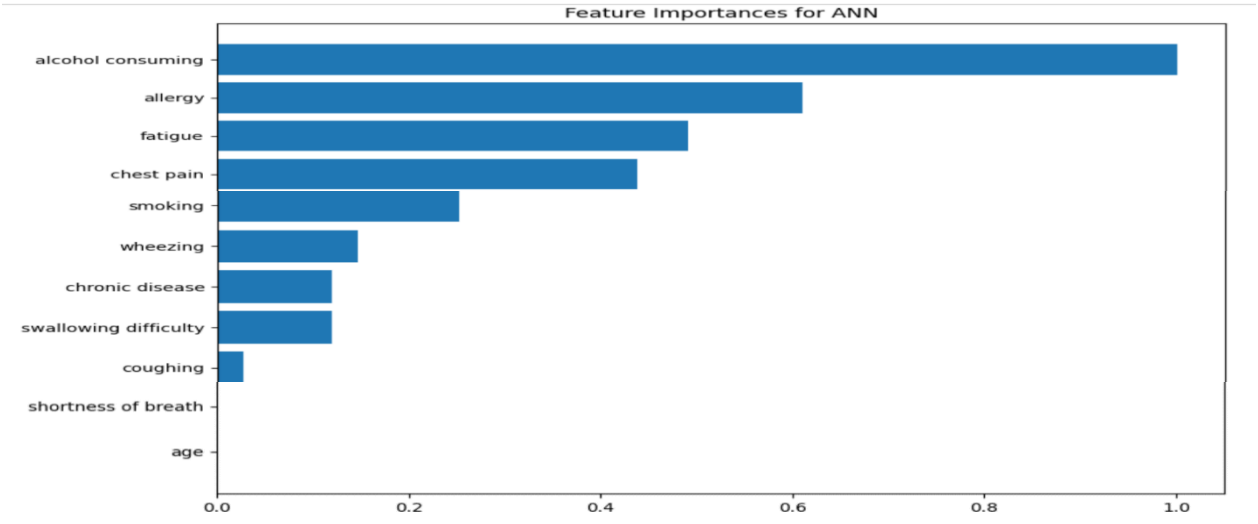

Fig 8: Feature Importance of ANN

- **Evaluating Model Performance on the Dataset Having Important Features:**

  Now after discarding the features having very low feature importance value from the concatenated dataset the model is again trained and tested with the new dataset formed after discarding the feature coefficient value.

- **Hyperparameter Tuning:**

  Hyperparameter tuning is the process of optimizing a machine learning model's external parameter (e.g., learning rate, number of layers) to improve performance, typically through methods like grid search or random search.
  Here, after tuning the hyperparameters the performance of the model is checked on the dataset having only high feature importance value. The accuracy is 95.53%.

3.6.4 Logistic Regression

Logistic Regression is a commonly used statistical technique for binary classification tasks. It estimates the probability of a binary outcome using one or more predictor variables. Here's a breakdown of the mathematical underpinnings and theory behind logistic regression, specifically applied to predicting lung cancer using clinical attributes:

- **Mathematical Formulation:**

- **Logistic Function:**

The core of logistic regression is the logistic (sigmoid) function, which maps any real-valued number into the range [0, 1]. The logistic function is defined as:

$$\sigma(z)=1/(1+e^{-z}) \quad \text{----------(4)}$$

- where z is a linear blend of the input features.

- **Model Equation:**

In logistic regression, the probability of the positive class (e.g., having lung cancer) is modeled as:

$$P(Y=1|X) =\sigma(w^{T}X+b) \quad \text{----------(5)}$$

where:

- X is the vector of input features (e.g., clinical attributes like age, smoking history, etc.).
- w is the vector of weights (coefficients) for the features.
- b is the bias term.
- σ is the sigmoid function.

The term $P(Y=1|X) =\sigma(w^{T}X+b)$ is the linear combination of the features plus the bias, and it serves as the input to the sigmoid function.

- **Probability and Odds:**

The logistic function provides the probability of the positive class, but in terms of odds, the model can be expressed as:

$$\text{Odds}= P(Y=1|X)/1-P(Y=1|X) =e^{(w^{T}X+b)} \quad \text{----------(6)}$$

- **Log-Odds (Logit):**

The log-odds (also called the logit) is given by:

$$\text{Logit}(P)=\log(P/(1-p) \quad \text{----------(7)}$$

- **Cost Function:**

  In logistic regression, the cost function (also known as the loss function) measures how well the model's predictions align with the actual outcomes. The objective is to reduce this cost function during the training process. For binary classification, such as predicting lung cancer (‘yes’/ ‘no’), the cost function is based on the logistic loss (also known as binary cross-entropy loss).

  Here’s a step-by-step derivation of the cost function for logistic regression:

- **Logistic Function (Sigmoid Function):**

  Logistic regression uses the logistic function to model the probability that a given instance belongs to the positive class (e.g., lung cancer = ‘yes’) or not. The logistic function is defined as:

$$h_\theta(x)=\sigma(z)= \frac{1}{1+e^{-z}} \text{----------(8)}$$

  where $z=\theta^T x$. Here, θ are the parameters of the model, and x is the feature vector.

- **Cost Function (Binary Cross-Entropy Loss):**

  The cost function measures the error between the predicted probabilities and the actual class labels. For a single training example, the cost function is:

$$\text{Cost}(h_\theta(x), y) = -\, y \log(h_\theta(x)) - (1-y) \log(1-h_\theta(x)) \text{----------(9)}$$

  This cost function penalizes incorrect predictions more heavily as they deviate further from the actual class labels.

- **Total Cost Function:**

  For the entire training set, the total cost function J(θ) is the average of the individual costs over all training examples:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} [-\, y^i \log(h_\theta(x^i)) \; - \; (1 - y^i) \log(1 - h_\theta(x^i))] \text{----------(10)}$$

  where:

  - m is the number of training examples.
  - y^i is the actual label of the i^{th} training example.
  - x^i represents the feature vector of the i^{th} training example.

- **Summary of Cost Function:**

The cost function for logistic regression in the context of predicting lung cancer (yes/no) is:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}[-y^i \log(\frac{1}{1+e^{-\theta^T x^i}}) - (1-y^i)\log(1-\frac{1}{1+e^{-\theta^T x^i}})$$ ----------(11)

This function penalizes the model for making incorrect predictions and is used to guide the learning process to achieve the best possible parameter estimates.

3.6.4.1 Applying Logistic Regression to Predict Lung Cancer

When predicting lung cancer, the dataset typically includes clinical attributes such as age, smoking, coughing, shortness of breath, alcohol consuming, swallowing difficulty, wheezing, chronic disease, fatigue, allergy, and chest pain. These features (attributes) are used as inputs to the logistic regression model. The model estimates the coefficients w for these features, by which the importance and effect size of each feature on the probability of having lung cancer is denoted**.**

- **Training:**

Here, in the process of training the logistic regression model to predict lung cancer 80% of the concatenated pre-processed labelled data is fed to the logistic regression model.

- **Testing:**

In this process the logistic regression model is given with the rest of the 20% unseen concatenated pre-processed unlabelled dataset to check the performance of the model while predicting lung cancer in patients.

- **Prediction:**

After applying logistic regression on the given unseen test dataset, the probability of having lung cancer for each patient is calculated by the model using the logistic (sigmoid) function. If, the value of probability for any instance is greater than or equal to the threshold (0.5) then the corresponding instance is classified to positive label (Lung Cancer= ‘yes’), otherwise it is classified to negative class (Lung Cancer= ‘yes’).

- **Feature Importance:**

In this stage the feature importance coefficient value of each attribute is measured for logistic regression. Then by using the data visualization technique the feature importance of all the features like age, smoking, coughing, shortness of breath, alcohol consuming, swallowing difficulty, wheezing, chronic disease, fatigue, allergy, and chest pain. Here, the high value of

the coefficient refers that the corresponding attribute highly related to the target outcome, i.e., lung cancer.

Example: The feature importance coefficient value of fatigue is greater than allergy for this logistic regression model. By this statement, it is suggested that there is a higher possibility of having lung cancer in a patient if he/she has been diagnosed with fatigue instead of allergy.

Now, after understanding the feature importance of each attribute the features having very low feature importance coefficient value is discarded from the oversampled concatenated dataset. Then the performance of the logistic regression model is again evaluated on the dataset having only the most important features.

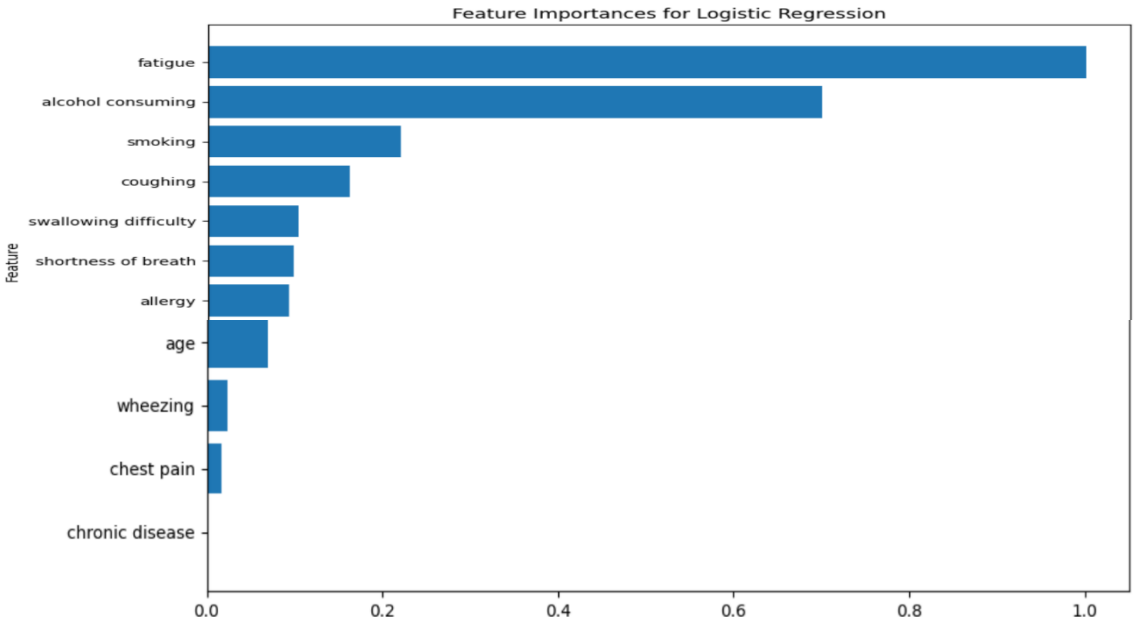

Fig 9: Feature Importance of Logistic Regression

- **Estimation of Lung Cancer Probability for Individual Patients:**

Now, it's time to calculate the probability of having lung cancer for each individual patient by using the logistic regression model. In order to do that some algorithm is used by the model to measure the probability of having lung cancer for each patient.

Now, the value of these probability is compared with the actual value of the target variable of the concatenated dataset to check if the model is accurate while making prediction about the presence of lung cancer on the test data. The threshold value of the probability is 0.5 means if the value of the probability of having lung cancer is greater than or equal to 0.5 then it is considered that the patient has lung cancer otherwise the patient does not have lung cancer.

- **Hyperparameter Tuning:**

In machine learning, hyperparameter tuning involves choosing the best set of hyperparameters for a model to optimize its performance. Hyperparameters are the configuration settings that are not learned from the data but set before the training process begins.

The hyperparameters used in logistic regression are:
{'C': 1.0,
'class_weight': None,
'dual': False,
'fit_intercept': True,
'intercept_scaling': 1,
'l1_ratio': None,
'max_iter': 100,
'multi_class': 'auto',
'n_jobs': None,
'penalty': 'l2',
'random_state': None,
'solver': 'lbfgs',
'tol': 0.0001,
'verbose': 0,
'warm_start': False}

The above-mentioned value of these hyperparameters is used before the process of hyperparameter tuning. In hyperparameter tuning the value of some of the hyperparameters are changed for the new dataset having only the features with high feature importance coefficient value.
Example: 'penalty': ['l1', 'l2'], 'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000],
'solver': ['liblinear', 'saga'].

The accuracy of the logistic regression model after hyperparameter tuning is 98%.
Also, with this tuned model the probability of having lung cancer for individual patient is measured and compared the target outcome with actual test set.

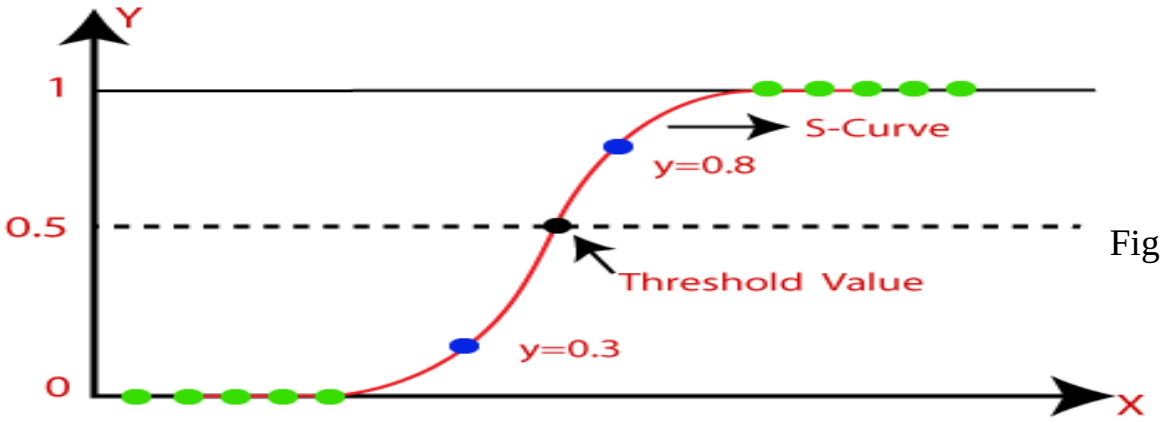

Fig:

Fig10: Logistic Regression

Source: https://www.javatpoint.com/logistic-regression-in-machine-learning

3.6.5 Support Vector Machine

SVM stands for Support Vector Machine. It is a supervised machine learning algorithm used for classification and regression tasks. The main idea behind SVM is to find a hyperplane that best separates the data into different classes. When the data is not linearly separable, SVM employs a technique known as the kernel trick to map the data into a higher-dimensional space, making it separable with a hyperplane.

The key components of Support Vector Machine (SVM) include:

- **Hyperplane:**

  A decision boundary that separates different classes in the lung cancer dataset (‘yes’, ‘no’) in the feature space. In n-dimensional space, a hyperplane is an (n-1)-dimensional subspace.

- **Support Vectors:**

  Data points that are closest to the hyperplane. These points are crucial in determining the position and orientation of the hyperplane. They are used to maximize the margin between different classes present in the concatenated lung cancer dataset (‘yes’, and ‘no’).

- **Margin:**

  The distance between the hyperplane and the closest data point from each class (‘Yes’ and ‘No’). SVM aims to maximize this margin to achieve better generalization on unseen data.

- **Kernel Trick:**

  A method used to transform the original feature space into a higher-dimensional space where a linear separation is possible. Common kernel functions include:

  - Linear Kernel: $\langle x, y \rangle$
  - Polynomial Kernel: $(\langle x, y \rangle + c)^d$
  - Radial Basis Function (RBF) Kernel: $\exp(-\gamma \|x-y\|^2)$
  - Sigmoid Kernel: $\tanh(\alpha \langle x, y \rangle + c)$

  In order to separate the classes ('YES' and ''NO') in the lung cancer dataset the linear kernel is used as:

  - In high dimensional feature space, the lung cancer dataset is linearly separable with a linear kernel.

  - Linear SVMs are computationally less expensive compared to non-linear kernels. They are faster to train, especially on large datasets with many features like as the lung cancer dataset which is used in this research work.
  - SVM with linear kernel function is less prone to overfitting compared to more complex non-linear models. This can help in achieving better generalization on unseen data.

- **Cost Parameter (C)**:

  A regularization parameter that balances the trade-off between expanding the margin and reducing classification error. A higher value of C emphasizes a lower classification error, while a lower value allows for a larger margin.

- **Cost Function:**

  The cost function for the SVM in this context integrates the goals of maximizing the margin and minimizing the classification error. The mathematical form of cost function used here is:

$$\frac{1}{2}||w||^2 + C\sum_{i=1}^{n}\xi_i \text{ ----------(12)}$$

  Here's a breakdown of the components:

- $\frac{1}{2}||w||^2$: This term represents the margin maximization objective. It aims to find a hyperplane (defined by the weight vector w) that maximizes the distance to the nearest data points of each class.

- C: The regularization parameter that balances the trade-off between margin maximization and classification error minimization. A higher C value increases the focus on reducing classification errors.

- ξ_i: Slack variables that allow some data points to be within the margin or on the wrong side of the hyperplane. These variables account for misclassifications and make the SVM more robust to noisy data.

3.6.5.1 Working Principal of SVM to Predict Lung Cancer:

➢ **Training:**

80% of the concatenated dataset is used as the training test for training the SVM model.

➢ **Testing:**

In this process the SVM model is given with the rest of the 20% concatenated pre-processed unlabelled dataset to check the performance of the model while making prediction of lung cancer in patients.

➢ **Prediction:**

In an SVM model with a linear kernel, the decision boundary is a hyperplane that separates the two classes. This hyperplane is defined in such a way that it maximizes the margin between the two classes, which is the distance between the hyperplane and the nearest data point of each class.

➢ **Hyperplane Equation**:

- For a linear kernel SVM, the decision boundary can be described by the equation $f(x) = w.x+b$, where w is the weight vector, x is the feature vector, and b is the bias term.

➢ **Classification Rule**:

- An instance x is classified as positive (lung cancer = 'yes') if $w \cdot x+b \geq 0$.
- An instance x is classified as negative (lung cancer = 'yes') if $w \cdot x+b<0$.
- The ‘predict’ method computes the decision function for each instance in concatenated scaled test set and assigns class labels based on the computed scores.

➢ **Feature Importance:**

In this stage the feature importance coefficient value of each attribute is measured for the SVM classifier. Then by using data visualization technique the feature importance of all the features.

The feature importance coefficient value of coughing is greater than alcohol consuming for this SVM. By this statement, it is suggested that there is a higher possibility of having lung cancer in a patient if he/she has diagnosed with coughing instead of alcohol consuming.

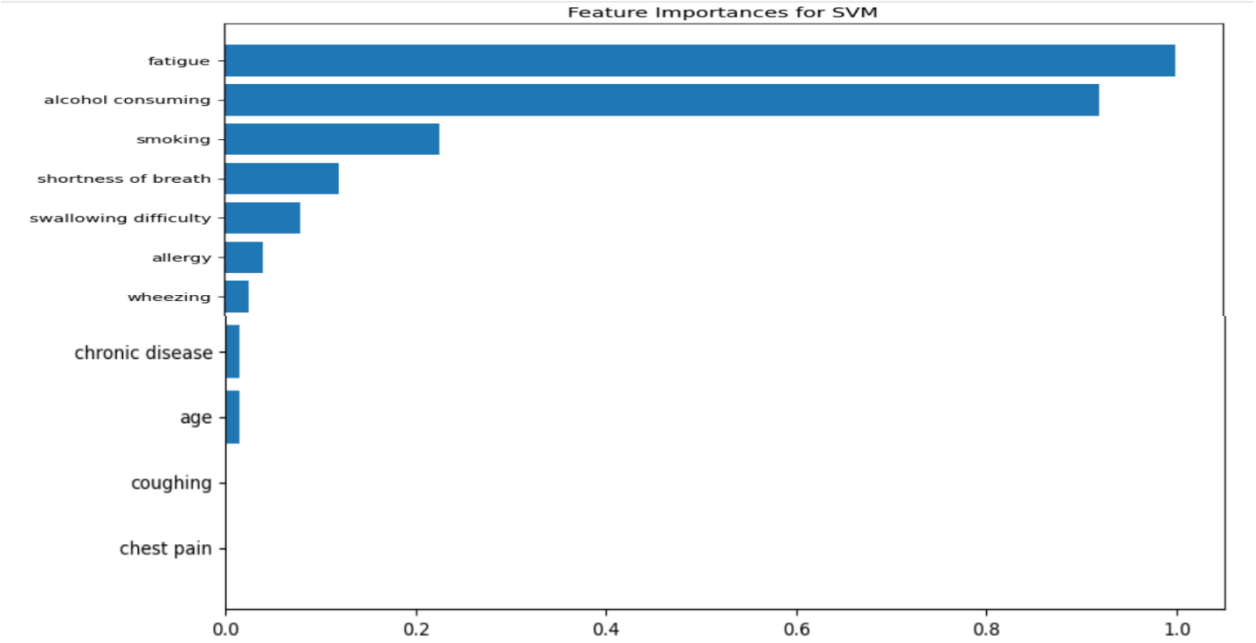

Fig11: Feature Importance of SVM

Now, after understanding the feature importance of each attribute the features having very low feature importance coefficient value is discarded from the concatenated pre-processed dataset. Then, with the new dataset having only the most important attributes having high feature importance coefficient value the SVM is trained and tested.

- **Estimation of Lung Cancer Probability for Individual Patients Using SVM:**

Now, it's time to calculate the probability of having lung cancer for each individual patient by using the SVM model. In order to do that certain algorithm is used by the model to measure the probability of having lung cancer for each patient.

Now, the value of these probability is compared with the actual value of the target variable of the concatenated pre-processed dataset to check if the model is accurate while making prediction about the presence of lung cancer on the test data. The threshold value of the probability is 0.5 means if the value of probability of having lung cancer is greater than or equal to 0.5 then it is considered that the patient has lung cancer otherwise the patient does not have lung cancer.

Example: The value of probability of having lung cancer in the first patient used in the test set for the SVM is 0.83 by which it is predicted that the patient does has a high probability of having lung cancer and after checking the actual target outcome for the same corresponding patient in test set it is found that the value of the actual target outcome is yes.

- **Hyperparameter Tuning for SVM:**

In machine learning, hyperparameter tuning involves selecting the best set of hyperparameters to optimize model performance. Hyperparameters are the settings that are predetermined before training starts and are not derived from the data.
Here the used hyperparameters before tuning the model are:

```
(  kernel='linear',
   C=1,
   probability=True,
   random_state=42,
   tol=1e-4,
   max_iter=-1,
   class_weight=None,
   verbose=False,
   shrinking=True,
   decision_function_shape='ovr',
   break_ties=False
)
```

And after tuning the hyperparameters are:

```
  ( 'C': [0.1, 1, 10],
   'kernel': ['linear', 'rbf'],
   'tol': [1e-3, 1e-4],
   'shrinking': [True, False],
   'decision_function_shape': ['ovr', 'ovo']
 )
```

Now with this tuned model the probability the lung cancer prediction is done with the test accuracy of 97.39%.

Also, the model is used to calculate the probability of each individual patient probability of having lung cancer. And the value of the probability is compared with the target outcome of the actual test set.

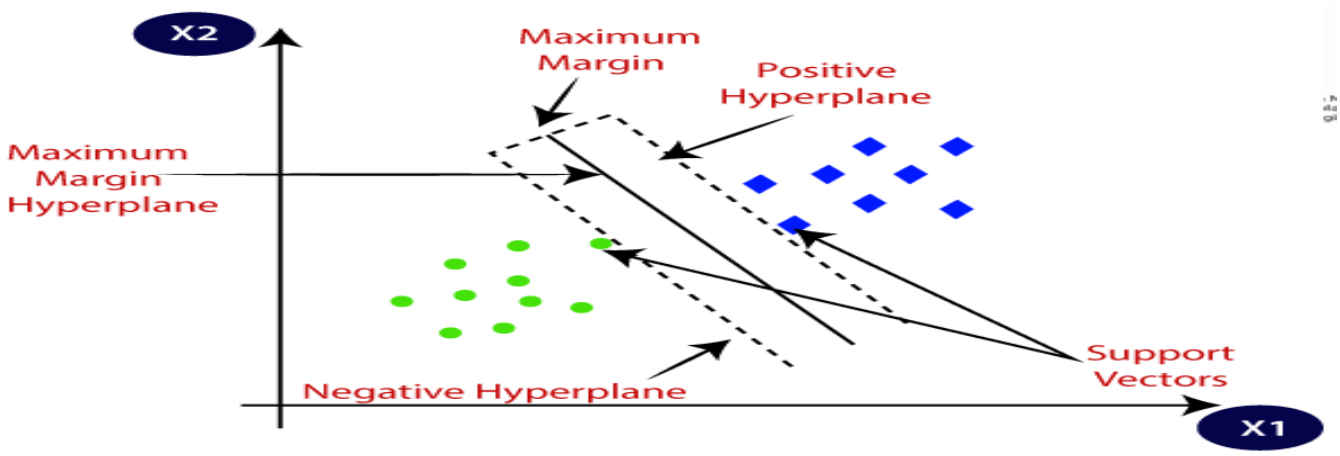

Fig12: Support Vector Machine

Source: https://www.javatpoint.com/machine-learning-support-vector-machine-algorithm

3.6.6 Gaussian Naive Bays Classifier

Gaussian Naive Bayes (GNB) is a variant of the Naive Bayes classifier that's specifically designed for continuous data. It assumes that the continuous features follow a Gaussian (normal) distribution. The fundamental principles of GNB are derived from Bayes' theorem, but it makes an additional assumption about the distribution of the feature values.

3.6.6.1 Key Concepts:

- **Bayes' Theorem**:

$$P(A|B) = \frac{P(B|A)\cdot P(A)}{P(B)} \text{ ----------(13)}$$

P(A|B) = Posterior probability of class A given predictor B.
$P(B|A)$= Likelihood, which is the probability of predictor B given class A.
P(A)= Prior probability of class A.
P(B)= Prior probability of predictor B.

- **Independence Assumption**: Each feature is assumed to contribute independently to the probability, simplifying the model's computation.

3.6.6.2 Mathematical Formulation:

- **Calculate Prior Probabilities:**

Calculate the prior probabilities P (C= lung Cancer[‘yes’]).

- **Calculate Mean and Variance:**

For each feature (e.g., age, smoking history), calculate the mean (μ_C) and variance (σ^2_C) for each patient.

- **Calculate Likelihood**:

For each feature value, compute the likelihood using the Gaussian distribution.
Likelihood $P(X_i|C)$ refers to the probability of observing the feature value X given that the patient has lung cancer.

$$P(X_i|C) = \frac{\mathbf{1}}{\sqrt{2\pi\sigma^2_{Ci}}} \exp\left(-\frac{(X_i-\mu_{Ci})\text{^}2}{2\sigma^2_{Ci}}\right) \text{----------(14)}$$

✓ Xi: Value of the i^{th} feature for a particular patient.

- ✓ μ_{Ci}: Mean of the i[th] feature for the class C (e.g., mean age of patients with lung cancer).
- ✓ σ_{Ci}: Standard deviation of the i[th] feature for class C.

- **Calculating the Likelihood for All Features:**

  Since Naive Bayes assumes that features are independent, the likelihood of observing all features X given the class C is the product of the individual likelihoods:

$$P(X|C) = P(X_1, X_2 \ldots, Xn|C) = \prod_{i=1}^{n} P(X_i \mid C) \text{----------(15)}$$

  where n is the total number of features.

- **Compute Posterior Probability**:
  Combine the prior probabilities and likelihoods to compute the posterior probability for lung cancer:

$$P(C=\text{Lung Cancer [yes]}|X) = \frac{P(X|C=\text{lung Cancer[yes]}) * P(C=\text{lung Cancer [yes]})}{P(X)} \text{----------(16)}$$

3.6.7 Working Principal of Naive Bays Classifier for Lung Cancer Prediction

- **Problem Context:**

  The dataset consists patient information, including features such as age, smoking, chronic disease, fatigue, allergy, wheezing, alcohol consuming, coughing, shortness of breath, swallowing difficulty, chest pain, and a label indicating whether the patient has lung cancer or not. The goal is to use the Gaussian Naive Bayes classifier to predict the probability that a given patient has lung cancer or not.

- **Training:**

  Previously, in the stage of data preprocessing some operations were performed on the concatenated dataset. Like as handling missing values, splitting the data into training and testing sets, encoding categorical variables, and scaling numerical features. These operations basically transform the raw, unusable data into an appropriate format that can be used in training and validating machine learning models. Here, in the process of training the Gaussian Naive Bayes Classifier to predict lung cancer 80% of the pre-processed data is fed to the model.

- **Testing:**

  In this process the Gaussian Naive Bayes classifier is given with the rest of the 20% unseen concatenated pre-processed test dataset to check the performance of the model while making prediction of lung cancer in patients.

- **Prediction:**

  After applying Gaussian Naive Bayes classifier on the given unseen test dataset, the probability of having lung cancer for each individual patient is calculated by the model. If the value of probability for any instance is greater than the threshold (0.5) then the corresponding instance is classified as positive label (lung cancer= 'yes'), otherwise it is classified as negative class (lung cancer= 'no').

- **Feature Importance:**

  In this stage of feature extraction, the feature importance coefficient value of each attribute is measured. Then by using data visualization technique the feature importance of all the features.

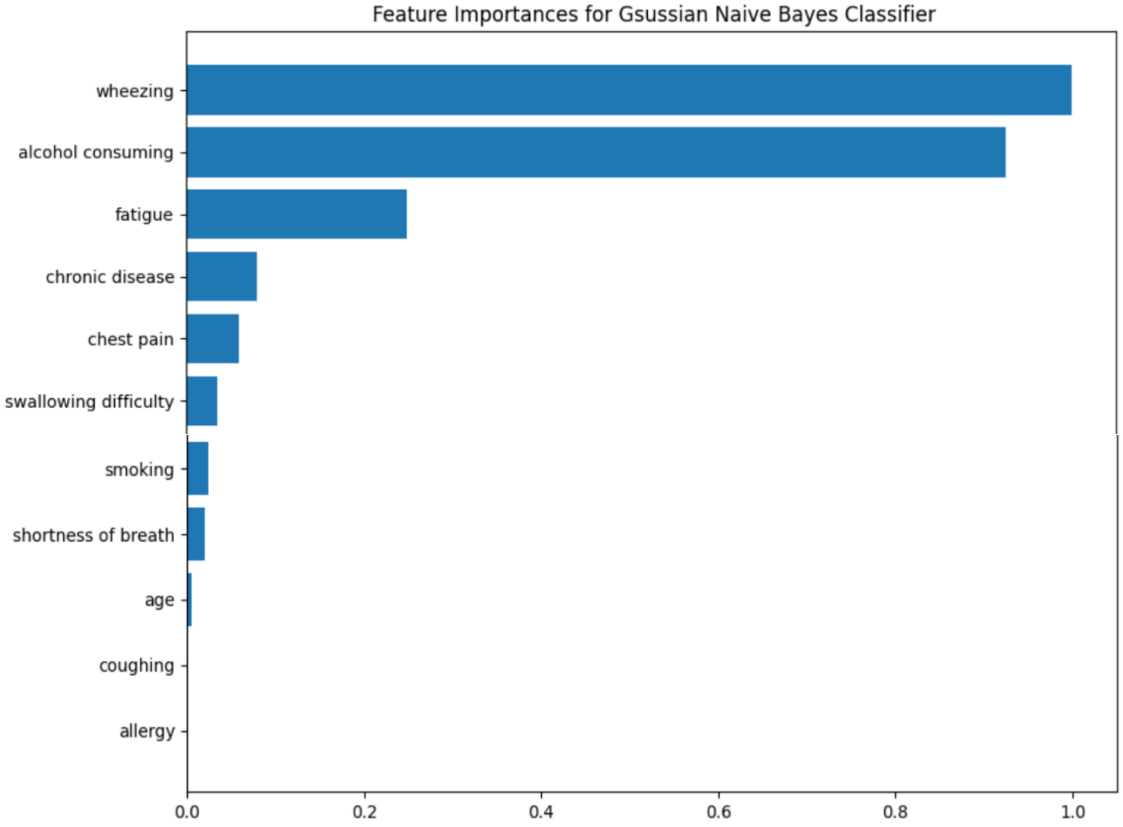

Fig13: Feature Importance for Gaussian Naïve Bayes Classifier

3.6.8 Estimation of Lung Cancer Probability for Individual Patients Using Gaussian Naive Bayes

Now, it's time to calculate the probability of having lung cancer for each individual patient by using the Gaussian Naive Bayes classifier model. In order to do that certain algorithm is used by the model to measure the probability of having lung cancer for each patient.

Now, the value of these probability is compared with the actual value of the target variable of the test dataset to check if the model is accurate while making prediction about the presence of lung cancer on the test data. The threshold value of the probability is 0.5 means if the value of the probability of having lung cancer is greater than 0.5 then it is considered that the patient has lung cancer otherwise the patient does not have lung cancer.

Example: The value of probability of having lung cancer in the third patient used in the test set for the Gaussian Naive Bayes classifier is 1.00 by which it is predicted that the patient has a high probability of lung cancer and after checking the actual target outcome for the same corresponding patient in the test set it is found that the value of the actual target outcome is 'yes'.

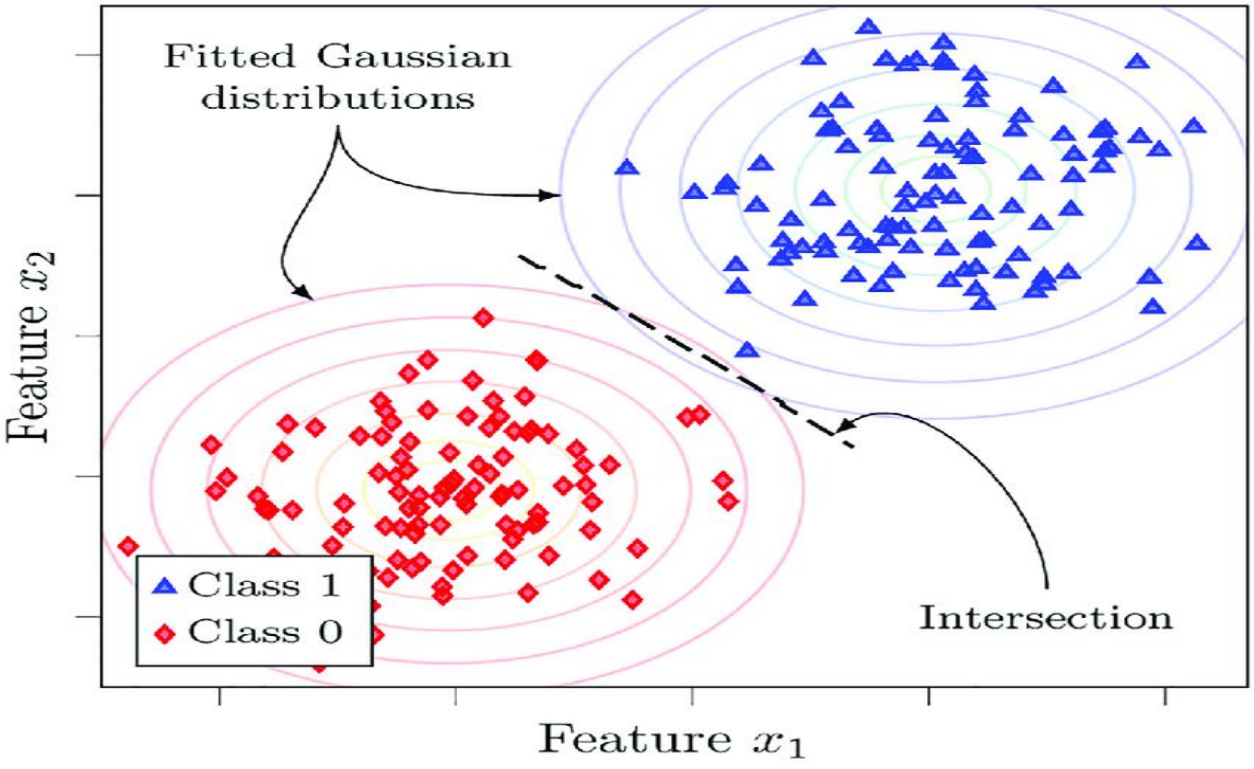

Fig14: Gaussian Naive Bayes Classifier

Source: https://rb.gy/nubzlt

CHAPTER 4

4. Experimentations And Results

The primary objective of this research is to develop and evaluate machine learning models for the early detection of lung cancer using patient data. Early and accurate detection of lung cancer can significantly improve patient outcomes and survival rates. To achieve this goal, a comprehensive dataset comprising patient medical records consisting of symptom-based attributes is utilized.

The data underwent extensive preprocessing, including cleaning, normalization, and feature selection, to ensure its suitability for machine learning algorithms. Several machine learning models like ANN, Logistic Regression, Support Vector Machine (SVM) and Gaussian Naive Bayes are used to predict the likelihood of lung cancer. The experiments conducted were crucial in assessing the performance of these models and identifying the most effective approach for accurate lung cancer prediction.

To evaluate the performance of the models several matrices are discovered:

4.1 Confusion Matrix:

A confusion matrix in machine learning is a table used to evaluate the performance of a classification algorithm. It summarizes the outcomes of predictions by showing the count of true positives (correctly predicted positive cases), true negatives (correctly predicted negative cases), false positives (incorrectly predicted positive cases), and false negatives (incorrectly predicted negative cases). This allows for a detailed assessment of the model's accuracy and types of errors.

		Predicted	
		Negative **(N)** -	Positive **(P)** +
Actual	Negative -	True Negative **(TN)**	**False Positive (FP) Type I Error**
	Positive +	**False Negative (FN) Type II Error**	True Positive **(TP)**

Fig15: Confusion Matrix

Source: https://www.sciencedirect.com/science/article/pii/S0040162523002123

- **True Negative (TN):**

  A true negative in the context of lung cancer detection occurs when a diagnostic test correctly identifies that a patient does not have lung cancer. The test result is negative (indicating the absence of cancer), and the patient truly is cancer-free.

- **False Positive (FP):**

  A false positive in the context of lung cancer detection occurs when a diagnostic test incorrectly indicates that a patient has lung cancer when they do not have the disease. In other words, the test results are positive (suggesting the presence of cancer), but the patient is cancer-free**.**

- **False Negative (FN):**

  A false negative in the context of lung cancer detection occurs when a diagnostic test incorrectly indicates that a patient does not have lung cancer when, in fact, they do have the disease. In this case, the test results are negative (suggesting the absence of cancer), but the patient has cancer.

- **True Positive (TP):**

  In the context of lung cancer detection, a true positive refers to a situation where a diagnostic test correctly identifies a patient who actually has lung cancer. Essentially, the test's result is positive (indicating the presence of cancer), and the patient truly does have the disease**.**

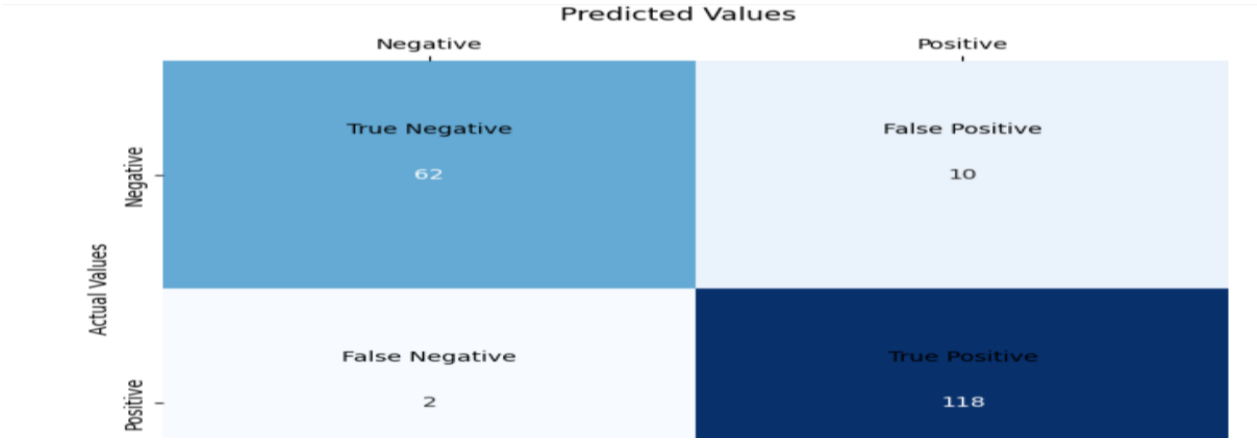

Fig16: Confusion Matrix Of ANN

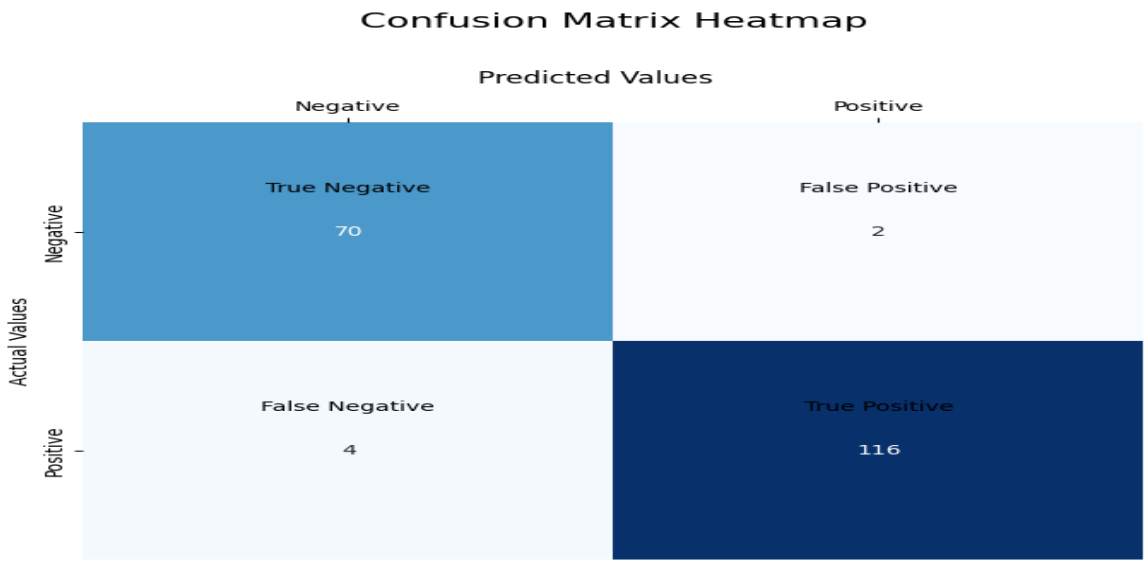

Fig17: Confusion Matrix of Logistic Regression

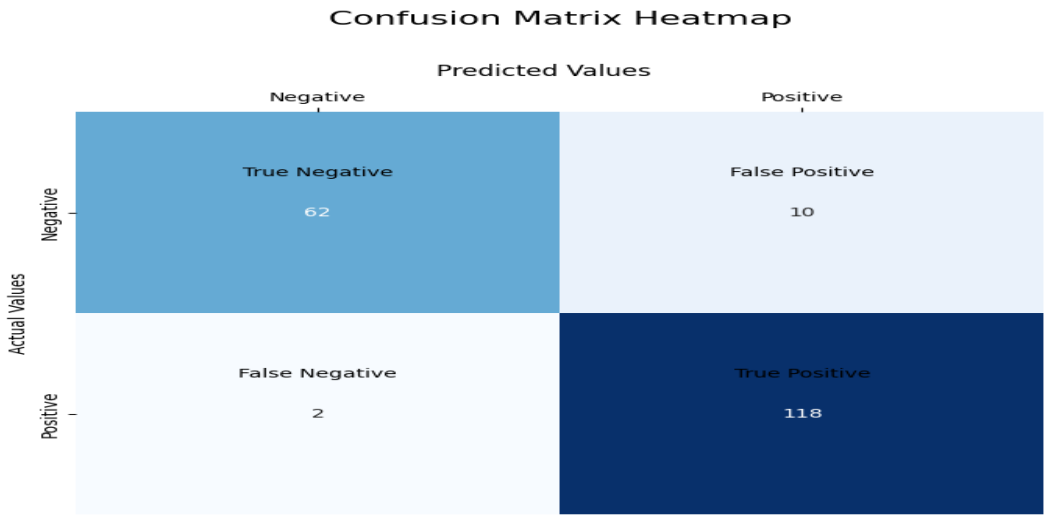

Fig18: Confusion Matrix of Naïve Bayes Classifier

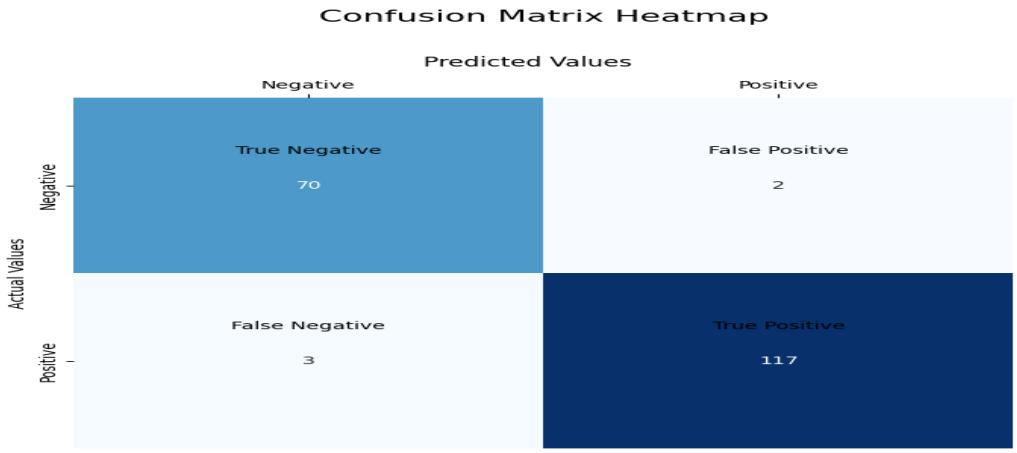

Fig19: Confusion Matrix of SVM

4.2 Performance Matrix

- **Accuracy:**

Measures the ratio of correctly classified instances to the total number of instances.

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN} \text{ ----------(17)}$$

- **Precision:**

Represents the proportion of positive identifications that were truly correct.

$$\text{Precision} = \frac{TP}{TP+FP} \text{ ----------(18)}$$

- **Recall:**

Measures the ratio of actual positives that were correctly identified.

$$\text{Recall} = \frac{TP}{TP+FN} \text{ ----------(19)}$$

- **F1 Score:**

The harmonic mean of precision and recall, provides a balance between the two.

$$\text{F1 score} = \frac{2*(Precision*Recall)}{(Precision+Recall)} \text{ ----------(20)}$$

- **Support:**

In the context of a performance matrix or classification report, support refers to the number of true instances (actual occurrences) of each class in the dataset. It essentially indicates how many samples of a particular class are present in the true labels.

- **Micro Average:**

  - Micro average summations the benefactions of all classes to cipher the average metric.
  - It counts the total true positives, false negatives, and false positives across all classes and then computes the precision, recall, or F1 score from these totals.

- **Weighted Average:**

  - Weighted averaging computes the metrics for each class individually and then averages them, weighting each class by its support (the number of true instances in that class).
  - The contribution of each class to the final average is proportional to the number of samples in that class.

4.3 Classification Report:

```
Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.88      0.93        72
           1       0.93      0.99      0.96       120

    accuracy                           0.95       192
   macro avg       0.96      0.93      0.94       192
weighted avg       0.95      0.95      0.95       192
```

Fig20: Classification Report for ANN

```
Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.88      0.93        72
           1       0.93      0.99      0.96       120

    accuracy                           0.95       192
   macro avg       0.96      0.93      0.94       192
weighted avg       0.95      0.95      0.95       192
```

Fig21: Classification Report for Logistic Regression

```
Classification Report for Naive Bayes Classifier:
              precision    recall  f1-score   support

           0       0.97      0.86      0.91        72
           1       0.92      0.98      0.95       120

    accuracy                           0.94       192
   macro avg       0.95      0.92      0.93       192
weighted avg       0.94      0.94      0.94       192
```

Fig22: Classification Report for Gaussian Naïve Bayes Classifier

Classification Report:

	precision	recall	f1-score	support
0	0.96	0.97	0.97	72
1	0.98	0.97	0.98	120
accuracy			0.97	192
macro avg	0.97	0.97	0.97	192
weighted avg	0.97	0.97	0.97	192

Fig23: Classification Report for SVM

CHAPTER 5

5. Comparative Analysis

Table 2: Classification Report of All Proposed Algorithms

	Model	Accuracy	Precision	Recall	F1 Score
0	ANN	0.96	0.94	0.94	0.94
1	Logistic Regression	0.97	0.97	0.97	0.97
2	Naive Bayes Classifier	0.94	0.94	0.94	0.94
3	SVM	0.97	0.97	0.97	0.97

classification report for all proposed models

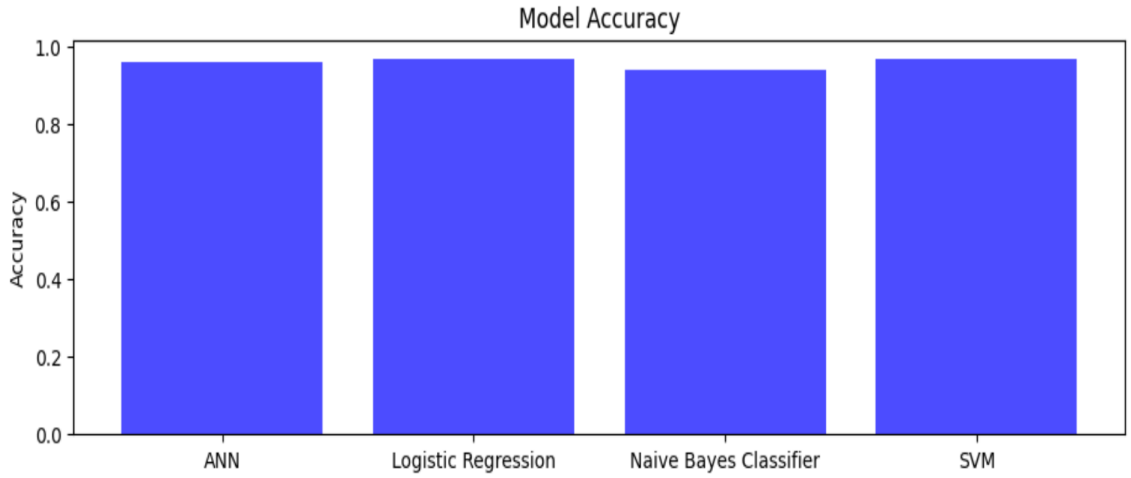

Fig 24: Comparison of Accuracy Using Bar Chart

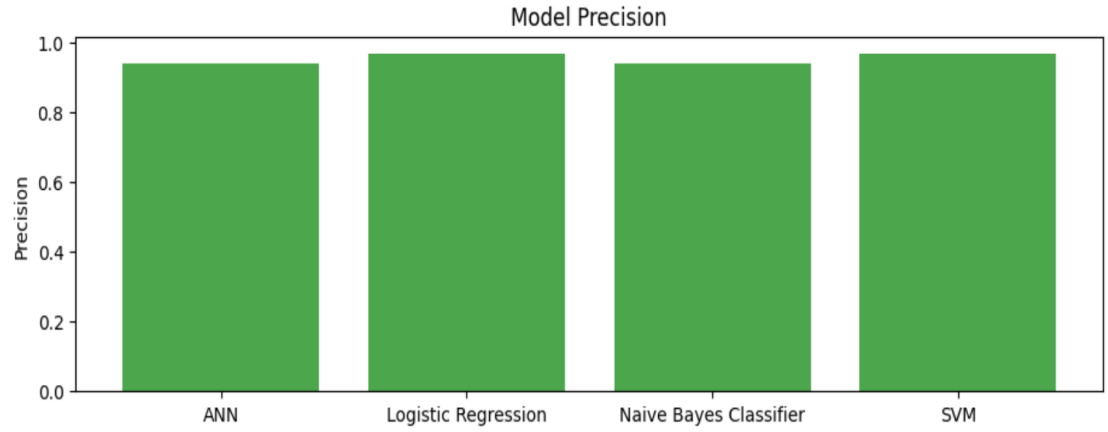

Fig 25: Comparison of Precision Using Bar Chart

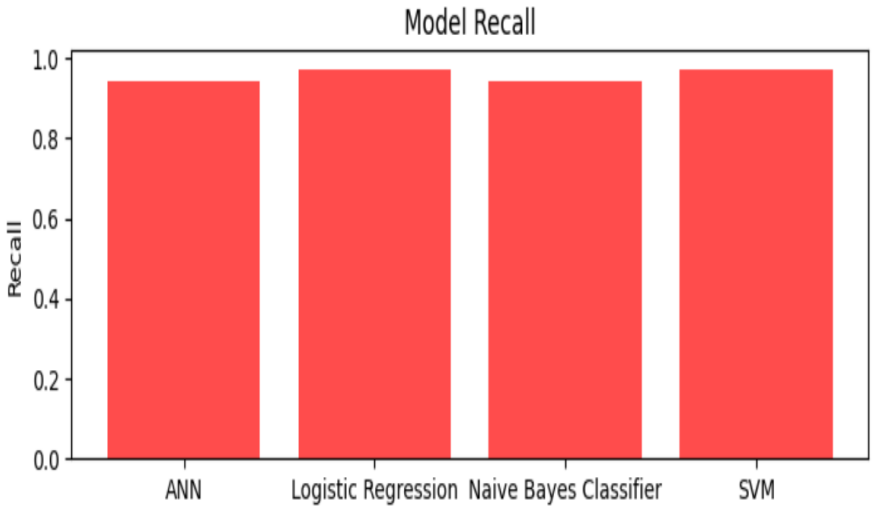

Fig 26: Comparison of Recall using Bar Chart

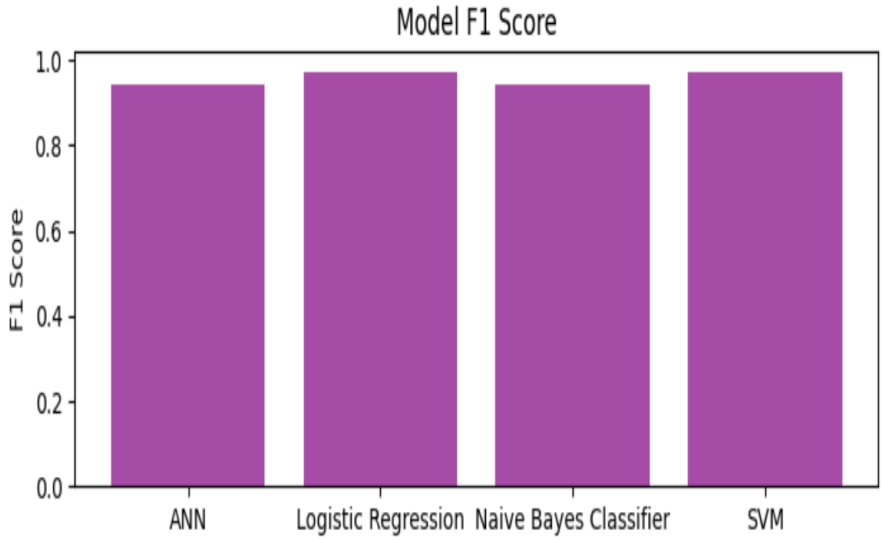

Fig 27: Comparison of F1 Score Using Bar Chart

CHAPTER 6

6. Conclusion And Future Scope

6.1 Conclusion

In this research work, a machine learning-based approach used to predict the presence of lung cancer in patients based on their medical history. By evaluating, the clinical attributes or symptoms the model makes the classification of the target variable, i.e., 'lung cancer' to either 'yes' or 'no. of having lung cancer. Four different models are used to do so. The models are ANN, Logistic Regression, Naïve Bayes Classifier, and SVM with accuracy of 96%,97%,94%, and 97% respectively. The model with the highest accuracy is Logistic Regression and SVM.
Each model is used to calculate the probability of having lung cancer and comparing it with the actual test set the appropriateness is checked for each model. Also, feature importance shows the relationship of having cancer and the attributes making it easier to understand the high-risk factors for lung cancer.

6.2 Future Scope

Here's a concise list of future research scopes of this research work:

- Multi-Modal Data Integration: Combine symptom-based data with imaging and genetic information.
- Real-Time Detection Systems: Develop tools for instant feedback and integration into clinical workflows.
- Explainable AI (XAI): Enhance model interpretability for better clinical decision-making.
- Cross-Disease Application: Apply methods to other cancers or chronic diseases.
- Longitudinal Data: Incorporate time-series data for improved risk prediction.
- Optimization and Scalability: Improve model performance and ensure broad applicability.
- Personalized Medicine: Tailor predictions and recommendations to individual patients.
- Collaborations and Data Sharing: Partner with institutions for better model validation and refinement.

REFERENCES

[1] M. Nasser and S. S. Abu-Naser, "Lung Cancer Detection Using Artificial Neural Network," International Journal of Engineering and Information Systems (IJEAIS), vol. 3, no. 3, pp. 17-23, Mar. 2019.

[2] Park, Sang Min, et al. "Impact of prediagnosis smoking, alcohol, obesity, and insulin resistance on survival in male cancer patients: National Health Insurance Corporation Study." Journal of clinical Oncology 24.31 (2006): 5017-5024.

[3] Xing, P-Y, Zhu, Y-X, Wang, L, et al. What are the clinical symptoms and physical signs for non-small cell lung cancer before diagnosis is made? A nation-wide multicenter 10-year retrospective study in China. Cancer Med. 2019; 8: 4055– 4069. https://doi.org/10.1002/cam4.2256

[4] Walter, F., Rubin, G., Bankhead, C. et al. Symptoms and other factors associated with time to diagnosis and stage of lung cancer: a prospective cohort study. Br J Cancer 112, S6–S13 (2015). https://doi.org/10.1038/bjc.2015.30

[5] V. Krishnaiah, G. Narsimha, and N. Subhash Chandra, "Diagnosis of Lung Cancer Prediction System Using Data Mining Classification Techniques," International Journal of Computer Science and Information Technologies (IJCSIT), vol. 4, no. 1, pp. 39–45, 2013.

[6] S. P. Maurya, P. S. Sisodia, R. Mishra, and D. P. Singh, "Performance of machine learning algorithms for lung cancer prediction: a comparative approach," *Scientific Reports*, vol. 14, no. 18562, 2024. DOI: 10.1038/s41598-024-58345-8.

[7] Divya, T., and J. Viji Gripsy. "AN INTEGRATED DEEP LEARNING BASED ENHANCED GREY WOLF OPTIMIZATION FOR LUNG CANCER PREDICTION." (2024).

[8] Levitsky, Adrian, et al. "Early symptoms and sensations as predictors of lung cancer: a machine learning multivariate model." *Scientific Reports* 9.1 (2019): 16504.

[9] Liao, Jia, et al. "Profiling symptom burden and its influencing factors at discharge for patients undergoing lung cancer surgery: a cross-sectional analysis." *Journal of Cardiothoracic Surgery* 17.1 (2022): 229.

[10] McCarty, Rachel D., et al. "Pathways to lung cancer diagnosis among individuals who did not receive lung cancer screening: a qualitative study." *BMC Primary Care* 24.1 (2023): 203.

[11] Devihosur, Nasareenbanu, and Ravi Kumar MG. "Enhancing Precision in Lung Cancer Diagnosis Through Machine Learning Algorithms." *International Journal of Advanced Computer Science and Applications* 14.8 (2023).

[12] Manju, B. R., V. Athira, and Athul Rajendran. "Efficient multi-level lung cancer prediction model using support vector machine classifier." *IOP Conference Series: Materials Science and Engineering*. Vol. 1012. No. 1. IOP Publishing, 2021.

[13] Ojha, Trailokya Raj. "Machine Learning based Classification and Detection of Lung Cancer." *Journal of Artificial Intelligence and Capsule Networks* 5.2 (2023): 110-128.

[14] Mukherjee, Swati, and S. U. Bohra. "Lung cancer disease diagnosis using machine learning approach." 2020 3rd International Conference on Intelligent Sustainable Systems (ICISS). IEEE, 2020.

[15] E. Dritsas and M. Trigka, "Lung Cancer Risk Prediction with Machine Learning Models," *Big Data and Cognitive Computing*, vol. 6, no. 4, p. 139, 2022. [Online]. Available: https://doi.org/10.3390/bdcc6040139.

[16] Bankar, Atharva, Kewal Padamwar, and Aditi Jahagirdar. "Symptom analysis using a machine learning approach for early stage lung cancer." *2020 3rd international conference on intelligent sustainable systems (ICISS)*. IEEE, 2020.

[17] Al-Bander, Baidaa, Yousra Ahmed Fadil, and Hussain Mahdi. "Multi-criteria decision support system for lung cancer prediction." *IOP Conference Series: Materials Science and Engineering*. Vol. 1076. No. 1. IOP Publishing, 2021.

[18] Martins, Sandro J., et al. "Lung cancer symptoms and pulse oximetry in the prognostic assessment of patients with lung cancer." *BMC cancer* 5 (2005): 1-6.

[19] Patra, Radhanath. "Prediction of lung cancer using machine learning classifier." *Computing Science, Communication and Security: First International Conference, COMS2 2020, Gujarat, India, March 26–27, 2020, Revised Selected Papers 1*. Springer Singapore, 2020.

[20] Gültepe, Yasemin. "Performance of Lung Cancer Prediction Methods Using Different Classification Algorithms." *Computers, Materials & Continua* 67.2 (2021).

[21] https://www.kaggle.com/code/godwinnani/lung-cancer-prediction-with-symptoms/input

[22] https://data.world/cancerdatahp/lung-cancer-data/workspace/file?filename=cancer+patient+data+sets.xlsx

APPENDIX

#Data Preprocessing
 # Exploratory Data Analysis

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

 # Importing the Dataset

```
df=pd.read_csv('B://OneDrive//Desktop//LUNG CACER THESIS
DATASET//LungCancer_dataset_kaggle.csv')
(df)

df.describe()

# Assuming the target variable is named 'target', replace it with the actual column name if
different
target_variable ='LUNG_CANCER'

# Count the occurrences of each class in the target variable
class_counts = df[target_variable].value_counts()

# Display the class proportions
print("Class counts:")
print(class_counts)
```

 # Null Value Checking

```
df.isnull()

#Outliers Checking Using Z-Score Method

import numpy as np
import pandas as pd

def detect_outliers(AGE):
    outliers = []
    threshold = 3
    mean = 62.705298
    std = np.std(AGE)  #std_dev = np.sqrt(np.sum((data - mean_value)^2) / len(data))

    for i in AGE:
        z_score = (float(i) - mean) / std  # Convert i to float
        if np.abs(z_score) > threshold:
            outliers.append(float(i))  # Convert i to float and append to outliers

    return outliers
```

```
# Assuming df is a DataFrame with a column named 'AGE'
df['AGE'] = pd.to_numeric(df['AGE'], errors='coerce')  # Convert 'AGE' column to numeric
outliers_list = detect_outliers(df['AGE'])
print(outliers_list)


import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Sample DataFrame

#dataframe = pd.DataFrame(AGE)

# Fetch values from the 'Values' column dynamically
column_of_interest = 'AGE'
values = df[column_of_interest].tolist()

# Function to detect outliers using Z-score
def detect_outliers_z_score(column, threshold=3):
    z_scores = np.abs((column - np.mean(column)) / np.std(column))
    outliers = z_scores > threshold
    return outliers

# Detect outliers in the specified column
outliers = detect_outliers_z_score(df[column_of_interest])

# Scatter plot with outliers highlighted
plt.scatter(df.index, df[column_of_interest], label='Data')
plt.scatter(df.index[outliers], df[column_of_interest][outliers], color='red', label='Outliers')
plt.xlabel('Index')
plt.ylabel(column_of_interest)
plt.title(f'Scatter Plot with Outliers in {column_of_interest}')
plt.legend()
plt.show()


#import numpy as np
import pandas as pd

def detect_outliers(ANXIETY):
    outliers = []
    threshold =3
    mean = 1.503311
    std = np.std(ANXIETY)  #std_dev = np.sqrt(np.sum((data - mean_value)^2) / len(data))

    for i in ANXIETY:
        z_score = (float(i) - mean) / std  # Convert i to float
        if np.abs(z_score) > threshold:
            outliers.append(float(i))  # Convert i to float and append to outliers
```

```
    return outliers

# Assuming df is a DataFrame with a column named 'AGE'
df['ANXIETY'] = pd.to_numeric(df['ANXIETY'], errors='coerce')  # Convert 'ANXIETY'
column to numeric
outliers_list = detect_outliers(df['ANXIETY'])
print(outliers_list)


import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Sample DataFrame

#dataframe = pd.DataFrame(ANXIETY)

# Fetch values from the 'Values' column dynamically
column_of_interest = 'ANXIETY'
values = df[column_of_interest].tolist()

# Function to detect outliers using Z-score
def detect_outliers_z_score(column, threshold=3):
    z_scores = np.abs((column - np.mean(column)) / np.std(column))
    outliers = z_scores > threshold
    return outliers

# Detect outliers in the specified column
outliers = detect_outliers_z_score(df[column_of_interest])

# Scatter plot with outliers highlighted
plt.scatter(df.index, df[column_of_interest], label='Data')
plt.scatter(df.index[outliers], df[column_of_interest][outliers], color='red', label='Outliers')
plt.xlabel('Index')
plt.ylabel(column_of_interest)
plt.title(f'Scatter Plot with Outliers in {column_of_interest}')
plt.legend()
plt.show()


#import numpy as np
import pandas as pd

def detect_outliers(SMOKING):
    outliers = []
    threshold =3
    mean = 1.562914
    std = np.std(SMOKING)  #std_dev = np.sqrt(np.sum((data - mean_value)^2) / len(data))

    for i in SMOKING:
        z_score = (float(i) - mean) / std  # Convert i to float
```

```
        if np.abs(z_score) > threshold:
            outliers.append(float(i))  # Convert i to float and append to outliers

    return outliers

# Assuming df is a DataFrame with a column named 'AGE'
df['SMOKING'] = pd.to_numeric(df['SMOKING'], errors='coerce')  # Convert 'ANXIETY'
column to numeric
outliers_list = detect_outliers(df['SMOKING'])
print(outliers_list)


import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Sample DataFrame

#dataframe = pd.DataFrame(SMOKING)

# Fetch values from the 'Values' column dynamically
column_of_interest = 'SMOKING'
values = df[column_of_interest].tolist()

# Function to detect outliers using Z-score
def detect_outliers_z_score(column, threshold=3):
    z_scores = np.abs((column - np.mean(column)) / np.std(column))
    outliers = z_scores > threshold
    return outliers

# Detect outliers in the specified column
outliers = detect_outliers_z_score(df[column_of_interest])

# Scatter plot with outliers highlighted
plt.scatter(df.index, df[column_of_interest], label='Data')
plt.scatter(df.index[outliers], df[column_of_interest][outliers], color='red', label='Outliers')
plt.xlabel('Index')
plt.ylabel(column_of_interest)
plt.title(f'Scatter Plot with Outliers in {column_of_interest}')
plt.legend()
plt.show()
row_numbers = df[df['AGE'] == 21].index.tolist()
print("Row numbers where AGE is 21:", row_numbers)
```

 # Outliers Removal

```
df=df.drop(22)   #This command deleted the row no '24' from the original dataset and stored the
final cleaned dataset as 'df'
```

```
df.head(30)

# Assuming your dataset has features and a target variable (X and y)
x = df.drop('LUNG_CANCER', axis=1)  # Adjust 'LUNG_CANCER' to the actual column name
y = df['LUNG_CANCER']


# Split the data into training and testing sets

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.utils import resample

#I have already defined X and Y

# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

# Apply one-hot encoding if needed
x_encoded = pd.get_dummies(x_train)

# Assuming 'GENDER' is a categorical column in X
x_encoded_train = pd.get_dummies(x_train, columns=['GENDER'], drop_first=True)
x_encoded_test = pd.get_dummies(x_test, columns=['GENDER'], drop_first=True)

# Check for any remaining non-numeric values after one-hot encoding
non_numeric_cols_train = x_encoded_train.select_dtypes(exclude=['float64', 'int64']).columns
non_numeric_cols_test = x_encoded_test.select_dtypes(exclude=['float64', 'int64']).columns

# Assuming 'target_column' is the target variable in Y
label_encoder = LabelEncoder()
y_encoded_train = label_encoder.fit_transform(y_train)
y_encoded_test = label_encoder.transform(y_test)

# Standardize the features
scaler = StandardScaler()
x_scaled_train = scaler.fit_transform(x_encoded_train)
x_scaled_test = scaler.transform(x_encoded_test)

print(x_train)
```

#Lung Cancer Detection Using ANN for the Original Dataset

```
from tensorflow.keras.layers import Input, Dense, Dropout, BatchNormalization
from tensorflow.keras.models import Sequential
import matplotlib.pyplot as plt
import tensorflow as tf

# Assuming you have x_train, x_test, y_encoded_train, y_encoded_test defined
```

```
# Define the number of features
num_features = x_train.shape[1]

# Define the architecture of the ANN with dropout regularization
model = Sequential([
    Input(shape=(num_features,)),
    Dense(128, activation='relu'),
    BatchNormalization(),
    Dropout(0.3),  # Adjusted dropout rate
    Dense(64, activation='relu'),
    BatchNormalization(),
    Dropout(0.3),  # Adjusted dropout rate
    Dense(32, activation='relu'),
    BatchNormalization(),
    Dropout(0.3),  # Adjusted dropout rate
    Dense(1, activation='sigmoid')
])

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model on the training data
history = model.fit(x_scaled_train, y_encoded_train, epochs=10, batch_size=64,
validation_split=0.2)


# Set random seeds for reproducibility
np.random.seed(42)
tf.random.set_seed(42)

# Evaluate the model on test data
test_loss, test_accuracy = model.evaluate(x_scaled_test, y_encoded_test, verbose=0)
print("Test Accuracy:", test_accuracy)

# Plot training history

plt.plot(history.history['val_accuracy'], label='Test Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()


df.head(30)
```

#Data concatination

#Importing the 2nd datset(from DataWorld)

```
df=pd.read_csv("F:\cancer patient data sets (2).csv")
print(df)
```

#Type Conversion

```
df['Dust Allergy'] = pd.to_numeric(df['Dust Allergy'], errors='coerce').fillna(0).astype('int64')

# Check the data type after conversion
print("After conversion:")
print(df['Dust Allergy'].dtype)

# Display the dataframe to verify the changes
print(df)
```

#Rename Needed Column

```
import pandas as pd
df = df.rename(columns={
   'Dust Allergy': 'Allergy',
   'chronic Lung Disease': 'Chronic Pain',
   'Dry Cough': 'Coughing',
   'Level': 'Lung Cancer',

})
df
```

#Visualization on the 1st (kaggle Dataset)

```
df2=pd.read_csv("B://OneDrive\Desktop//LUNG CACER THESIS
DATASET//LungCancer_dataset_kaggle.csv")
df2
```

#Deleting not matched columns from 2nd Dataset

```
import pandas as pd

# # Assuming df1 is your dataframe
columns_to_delete = ['OccuPational Hazards','Genetic Risk', 'Passive Smoker','Coughing of
Blood', 'Clubbing of Finger Nails','Frequent Cold','Snoring']
df = df.drop(columns=columns_to_delete)

# # Display the updated dataframe
print(df.head())
```

#Deleting recods having 'Medium' target value

```
import pandas as pd


# Drop rows where LUNG_CANCER is 'Medium'
df= df[df['Lung Cancer'] != 'Medium']

# Save the modified DataFrame back to a CSV if needed
df.to_csv('cleaned_dataset.csv', index=False)
df
```

#Rename 'Low' to 'No' and 'High' to 'Yes'

```
import pandas as pd
# Use .loc[] to avoid SettingWithCopyWarning
df.loc[:, 'Lung Cancer'] = df['Lung Cancer'].replace({'Low': 'No', 'High': 'Yes'})

# Save the modified DataFrame back to a CSV if needed
df.to_csv('modified_dataset.csv', index=False)
```

```
# Display the DataFrame to verify the changes
print(df)
```

#Encoding Categorical values

```
import pandas as pd

# Assuming df is your existing DataFrame

# List of columns to leave unchanged
columns_to_exclude = [ 'Patient Id',  'Age',   'Lung Cancer']

# Separate the columns to be excluded
df_excluded = df[columns_to_exclude].copy()

# Work on the remaining columns
df_remaining = df.drop(columns=columns_to_exclude).copy()

# Convert the selected columns to numeric values (if they are strings)
df_remaining = df_remaining.apply(pd.to_numeric, errors='ignore')

# Function to convert values
def convert_values(x):
    if pd.isna(x):  # Handle NaN values if any
        return x
    elif 1 <= x <= 4:
        return 1
    elif 5 <= x <= 9:
        return 2
    else:
        return x  # Leave the value unchanged if it doesn't fall in the range 1 to 9

# Apply the function to the remaining columns in the DataFrame
df_transformed = df_remaining.applymap(convert_values)

# Merge the excluded columns back into the transformed DataFrame
df_final = pd.concat([df_excluded, df_transformed], axis=1)
columns = list(df_final.columns)
columns.remove('Lung Cancer')  # Remove 'Lung Cancer' from its current position
columns.insert(18, 'Lung Cancer')  # Insert 'Lung Cancer' into the 17th position

# Reorder the DataFrame columns
df_final = df_final[columns]

# Now df_final has the 'Lung Cancer' column in the 17th position
# Now df_final has the transformed values along with the original excluded columns
print(df_final)
```

#Saving this preprocessed dataset in .csv format to a specified location for futher use

```
path =  r"B:\OneDrive\Desktop\LUNG CACER THESIS DATASET\cancer patients
datasets2.csv"
import os
# Save the DataFrame to the specified path
```

```
df_final.to_csv(path, index=False)


# Replace <YourUsername> with your actual username or use os to get the username dynamically
path = os.path.join(os.path.expanduser("~"), 'Desktop', 'cancer patients datasets1.csv')

# Save the DataFrame to the specified path
df_final.to_csv(path, index=False)
print(f"DataFrame saved to {path}")
```

#importaing preprocessed 2nd dataset

```
df1=pd.read_csv("B://OneDrive//Desktop//LUNG CACER THESIS DATASET//New folder (2)//cancer patient data sets final (2).csv")
df1
```

#importing the 1st datset

```
df2=pd.read_csv("B://OneDrive\Desktop//LUNG CACER THESIS DATASET//LungCancer_dataset_kaggle.csv")
df2
```

 # Data Preprocessing

#Deleting columns from df1 not present in the df2

```
import pandas as pd

# # Assuming df1 is your dataframe
columns_to_delete = ['Gender','Patient Id', 'Air Pollution', 'Balanced Diet', 'Obesity', 'Weight Loss']
df1 = df1.drop(columns=columns_to_delete)

# # Display the updated dataframe
print(df1.head())
```

#Ordering columns of df1 comaring with df2

```
import pandas as pd

# Rename columns in df3 to match the columns in df
df1.columns = ['AGE',  'ALCOHOL CONSUMING', 'ALLERGY', 'CHRONIC DISEASE', 'SMOKING', 'CHEST PAIN', 'FATIGUE', 'SHORTNESS OF BREATH', 'WHEEZING', 'SWALLOWING DIFFICULTY', 'COUGHING', 'LUNG_CANCER']

# Reorder the columns in df3 to match the order in df
df1 = df1[[ 'AGE', 'SMOKING', 'CHRONIC DISEASE', 'FATIGUE', 'ALLERGY', 'WHEEZING', 'ALCOHOL CONSUMING', 'COUGHING', 'SHORTNESS OF BREATH', 'SWALLOWING DIFFICULTY', 'CHEST PAIN', 'LUNG_CANCER']]
```

```
# Display the updated dataframe
print(df1.head())
```

#Type conversion of columns of df1 comparing with df2

```
df1['ALLERGY'] = pd.to_numeric(df1['ALLERGY'], errors='coerce').fillna(0).astype('int64')

# Check the data type after conversion
print("After conversion:")
print(df1['ALLERGY'].dtype)

# Display the dataframe to verify the changes
print(df1)

df2=pd.read_csv("B://OneDrive\Desktop//LUNG CACER THESIS
DATASET//LungCancer_dataset_kaggle.csv")
df2
```

#Deleting Columns from df2 comparing with df1

```
import pandas as pd

# Assuming df3 is your dataframe
columns_to_delete = ['GENDER','YELLOW_FINGERS','ANXIETY','PEER_PRESSURE']
df2 = df2.drop(columns=columns_to_delete)

# Display the updated dataframe
print(df2.head())
```

 # df2 having same columns with df1

```
df2
```

#df1 having same columns with df2

```
df1
```

#Concatination of df2 with df1

```
# Standardize column names in both DataFrames
df2.columns = df2.columns.str.lower().str.strip()
df1.columns = df1.columns.str.lower().str.strip()

# Now concatenate the DataFrames
df3 = pd.concat([df2, df1], ignore_index=True)
df3
```

#Typeconversion of Datatype for Concatenated Dataset

```
import pandas as pd

# Assuming df3 is your concatenated DataFrame
# Normalize the 'lung_cancer' column to have consistent casing
df3['lung_cancer'] = df3['lung_cancer'].str.strip().str.capitalize()

# Count the class occurrences again after normalization
class_counts_normalized = df3['lung_cancer'].value_counts()

# Print the normalized class counts
print(class_counts_normalized)
```

#Null value Checking of New Concatenated Dataset

```
# Check for null values in each column
null_values = df3.isnull().sum()

# Print the null value counts
print("\nNull values in each column:")
print(null_values)
```

#Dealing with Outliers

```
from scipy import stats
import numpy as np

# Dictionary to hold the count of outliers for each column
outlier_counts = {}

# Iterate through each numeric column in the DataFrame
for col in df3.select_dtypes(include=[np.number]).columns:
    # Calculate the Z-scores for the column
    z_scores = np.abs(stats.zscore(df3[col]))

    # Identify outliers (Z-score > 3)
    outliers = z_scores > 3

    # Count the number of outliers in the column
    outlier_count = np.sum(outliers)
    # Store the result in the dictionary
    outlier_counts[col] = outlier_count
# Convert the dictionary to a DataFrame for better readability
outlier_counts_df = pd.DataFrame(list(outlier_counts.items()), columns=['Column', 'Outlier
Count'])
# Print the outlier counts for each column
print("Outlier counts for each column based on Z-score (> 3):")
print(outlier_counts_df)
```

#Shuffling datapoints of df3

```
df3_shuffled = df3.sample(frac=1, random_state=42).reset_index(drop=True)

# Display the first few rows of the shuffled DataFrame
print("\nShuffled DataFrame:")
print(df3_shuffled.head())
```

#Feature Engineering

#Seperating independent features from target variable

```
# Assuming your dataset has features and a target variable (X and y)
X1 = df3_shuffled.drop('lung_cancer', axis=1)  # Adjust 'LUNG_CANCER' to the actual column name
Y1 = df3_shuffled['lung_cancer']
```

#Independent feature set data

```
X1
```

#Dependent featureset data

```
Y1
```

#Splitting dataset into training and testing

```
# Split the data into training and testing sets

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.utils import resample

#I have already defined X and Y
# Split the data into training and testing sets
X1_train, X1_test, Y1_train, Y1_test = train_test_split(X1, Y1, test_size=0.2, random_state=42)

#Independent Training set
X1_train
#Test set
X1_test
#Dependent train set
Y1_train
#Test set(Target Variable)
Y1_test

import pandas as pd
# Assuming Y1_test is a pandas Series
value_counts = pd.Series(Y1_test).value_counts()
```

```
print(value_counts)

# Encoding
 #Label Encoding

# Assuming 'target_column' is the target variable in Y
label_encoder = LabelEncoder()
Y1_encoded_train = label_encoder.fit_transform(Y1_train)
Y1_encoded_test = label_encoder.transform(Y1_test)


# Get the size of y_encoded_train
size_of_Y1_encoded_train = Y1_encoded_train.size

print("Size of y_encoded_train:", size_of_Y1_encoded_train)

#Y1_encoded_train
print(label_encoder.classes_)

#0=no,1=yes

from sklearn.preprocessing import LabelEncoder

# Define the classes
classes = ['Yes', 'No']

# Initialize and fit LabelEncoder
label_encoder = LabelEncoder()
label_encoder.fit(classes)

# Print the mapping
print("Class mapping:", dict(enumerate(label_encoder.classes_)))

X1_encoded_train=X1_train#we dont need to use one hot encoding as there is no categorical
value in the independen set
X1_encoded_test=X1_test



#  Scaling

# Standardize the features
scaler = StandardScaler()
X1_scaled_train = scaler.fit_transform(X1_encoded_train)
X1_scaled_test = scaler.transform(X1_encoded_test)


# Finding Correlation between Independent Attributes

import pandas as pd
import seaborn as sns
```

```
import matplotlib.pyplot as plt

# Assuming X_encoded is your DataFrame
# Calculate the correlation matrix
correlation_matrix = X1_encoded_train.corr()

# Print the correlation matrix
print("Correlation Matrix:\n", correlation_matrix)

# Visualize the correlation matrix using a heatmap
plt.figure(figsize=(14, 10))
sns.heatmap(correlation_matrix, annot=True, fmt='.2f', cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()

# Function to suggest feature combinations based on correlation threshold
def suggest_feature_combinations(corr_matrix, threshold=0.9):
    combinations = []
    for i in range(len(corr_matrix.columns)):
        for j in range(i + 1, len(corr_matrix.columns)):
            if abs(corr_matrix.iloc[i, j]) > threshold:
                combinations.append((corr_matrix.columns[i], corr_matrix.columns[j]))
    return combinations

# Suggest feature combinations based on a correlation threshold
threshold = 0.9  # You can adjust this threshold based on your needs
combinations = suggest_feature_combinations(correlation_matrix, threshold)

# Print the suggested feature combinations
print("Suggested Feature Combinations (based on correlation threshold of
{}):".format(threshold))


# Implementation of Machine Learning Algorithm for Lung Cancer Prediction

# Using ANN for Lung Cancer Prediction

from tensorflow.keras.layers import Dropout, Input, Dense
from tensorflow.keras.models import Sequential
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report  # Import this

# Assuming you have X1_encoded_train, X1_scaled_train, X1_scaled_test, Y1_encoded_train,
Y1_encoded_test defined

# Define the number of features
num_features = X1_encoded_train.shape[1]

# Define the model
model1 = Sequential([
    Input(shape=(num_features,)),
```

```
    Dense(16, activation='relu'),
    Dropout(0.2),
    Dense(16, activation='relu'),
    Dropout(0.2),
    Dense(1, activation='sigmoid')
])
# Compile the model
model1.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
# Train the model
history = model1.fit(X1_scaled_train, Y1_encoded_train, epochs=10, batch_size=64,
validation_split=0.2)
# Evaluate the model on the test set
test_loss, test_accuracy = model1.evaluate(X1_scaled_test, Y1_encoded_test, verbose=0)
print("Test Accuracy:", test_accuracy)
# Plot the accuracy over epochs
plt.plot(history.history['val_accuracy'], label='Test Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Accuracy')
plt.legend()
plt.show()
# Predict on the test set
Y_pred_prob = model1.predict(X1_scaled_test)
Y_pred = (Y_pred_prob > 0.5).astype("int32")
# Generate the classification report
report = classification_report(Y1_encoded_test, Y_pred)
print("Classification Report:\n", report)
plt.show()


#Visualization of Confusion Matrix for ANN using Heatmap

import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

# Generate predictions from the model
Y1_pred = model1.predict(X1_scaled_test)
Y1_pred_classes = (Y1_pred > 0.5).astype(int).reshape(-1)
# Compute the confusion matrix
cm = confusion_matrix(Y1_encoded_test, Y1_pred_classes)
# Plot the confusion matrix
plt.figure(figsize=(8, 6))
ax = sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False,
                 xticklabels=['Negative', 'Positive'],
                 yticklabels=['Negative', 'Positive'])
# Set the x-axis labels and move them to the top
ax.set_xticklabels(['Negative', 'Positive'])
ax.xaxis.set_ticks_position('top')
plt.xticks(rotation=0)
```

```
plt.ylabel('Actual Values')
plt.title('Confusion Matrix Heatmap', y=1.2, fontsize=16)
plt.figtext(0.5, 0.95, 'Predicted Values', ha='center', fontsize=12)
# Add text labels for true/false positives/negatives
labels = np.array([['True Negative ', 'False Positive'], ['False Negative', 'True Positive']])
for i in range(2):
    for j in range(2):
        ax.text(j + 0.5, i + 0.3, f"{labels[i, j]}", ha='center', va='center', color='black', fontsize=10)

plt.show()
```

#Visualizing ROC Curve for ANN

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc
# Step 1: Get predicted probabilities
Y1_pred_prob = model1.predict(X1_scaled_test).ravel()
# Step 2: Compute ROC curve
fpr, tpr, _ = roc_curve(Y1_encoded_test, Y1_pred_prob)
# Step 3: Compute AUC score
roc_auc = auc(fpr, tpr)
# Step 4: Plot ROC curve
plt.figure(figsize=(10, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (area = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='grey', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```

#Visualization of Feature Importance Coefficient using ANN

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.inspection import permutation_importance
from sklearn.base import BaseEstimator, ClassifierMixin
# Define a Keras model wrapper to be used with scikit-learn
class KerasClassifierWrapper(BaseEstimator, ClassifierMixin):
    def __init__(self, keras_model):
        self.keras_model = keras_model
        self.classes_ = np.array([0, 1])  # Update this based on your classification labels
    def fit(self, X, y):
        # This method is required but not used
        pass
    def predict(self, X):
        # Convert probabilities to binary labels
        return (self.keras_model.predict(X) > 0.5).astype("int32")
```

```
# Create an instance of the wrapper with your trained Keras model
model_wrapper = KerasClassifierWrapper(model1)
# Assuming you have a list of feature names
feature_names = [ 'age'          ,'smoking',      'chronic disease',        'fatigue',          'allergy',
          'wheezing',      'alcohol consuming',   'coughing',       'shortness of breath',
              'swallowing difficulty',      'chest pain']
# Calculate permutation feature importance
perm_importance = permutation_importance(model_wrapper, X1_scaled_test, Y1_encoded_test,
n_repeats=10, random_state=42, scoring='accuracy')
# Get the feature importances
feature_importances = np.abs(perm_importance.importances_mean)  # Take absolute values
# Normalize the feature importances to be within the range of 0 to 1
feature_importances_normalized = (feature_importances - feature_importances.min()) /
(feature_importances.max() - feature_importances.min())
# Print the normalized feature importances with their names
for name, importance in zip(feature_names, feature_importances_normalized):
    print(f"{name}: {importance:.4f}")
# Sort the feature importances in descending order
sorted_idx = np.argsort(feature_importances_normalized)[-15:]
# Plot the normalized feature importances
plt.figure(figsize=(10, 8))
plt.barh(range(len(sorted_idx)), feature_importances_normalized[sorted_idx])
plt.yticks(range(len(sorted_idx)), np.array(feature_names)[sorted_idx])
plt.title("Feature Importances for ANN")
plt.show()


Y2=Y1
print(Y2)
```

#Discarding Features with Negligible Feature Importance Coefficient Value(Thresold>=0.1)

```
import pandas as pd
# Assuming X is your DataFrame
print("Original DataFrame:")
print(X1.head())
# Columns to be deleted
columns_to_drop = ['chronic disease']
# Ensure the columns to drop are present in the DataFrame
columns_to_drop = [col for col in columns_to_drop if col in X1.columns]
# Drop the specified columns
X2 = X1.drop(columns=columns_to_drop)#saving the the datset without low feature column to
#new datset

print("DataFrame after dropping specified columns:")
print(X2.head())
```

#Pre-processing the New Dataset Having Only the Most Important Attributes

```
# Split the data into training and testing sets

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.utils import resample

Y2=Y1# In order to save the original datset saving the dependedent set to Y2

#I have already defined X and Y

# Split the data into training and testing sets
X2_train, X2_test, Y2_train, Y2_test = train_test_split(X2, Y2, test_size=0.2, random_state=42)
# Assuming 'target_column' is the target variable in Y
label_encoder = LabelEncoder()
Y2_encoded_train = label_encoder.fit_transform(Y2_train)
Y2_encoded_test = label_encoder.transform(Y2_test)
# Standardize the features
X2_encoded_train=X2_train# we dont nee one hot encoding as it is already in encoded from
X2_encoded_test=X2_test
scaler = StandardScaler()
X2_scaled_train = scaler.fit_transform(X2_encoded_train)
X2_scaled_test = scaler.transform(X2_encoded_test)

print(X2_encoded_train)
```

#Evaluating Performance of the ANN with the Dataset Consisting Only the Most Important Attributes

```
# # Import necessary libraries
from tensorflow.keras.layers import Dropout, Input, Dense
from tensorflow.keras.models import Sequential
from sklearn.metrics import roc_curve, accuracy_score, classification_report, confusion_matrix,
auc
import numpy as np
import matplotlib.pyplot as plt
# Assuming you have the following data prepared:
# X_scaled_train, X_scaled_test, Y_encoded_train, Y_encoded_test
# Define the ANN model
num_features = X2_encoded_train.shape[1]
model2 = Sequential([
    Input(shape=(num_features,)),
    Dense(16, activation='relu'),
    Dropout(0.2),
    Dense(16, activation='relu'),
    Dropout(0.2),
    Dense(1, activation='sigmoid')
])

# Compile the model
model2.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

```
# Train the model
history = model2.fit(X2_scaled_train, Y2_encoded_train, epochs=10, batch_size=64,
validation_split=0.2)

# Make predictions on the test set
Y2_pred_ann = model2.predict(X2_scaled_test)
Y2_pred_ann_classes = (Y2_pred_ann > 0.5).astype(int)

# Evaluate the model's performance
accuracy_ann = accuracy_score(Y2_encoded_test, Y2_pred_ann_classes)
classification_report_result_ann = classification_report(Y2_encoded_test, Y2_pred_ann_classes)
confusion_matrix_result_ann = confusion_matrix(Y2_encoded_test, Y2_pred_ann_classes)
fpr_ann, tpr_ann, thresholds_ann = roc_curve(Y2_encoded_test, Y2_pred_ann)
roc_auc_ann = auc(fpr_ann, tpr_ann)

# Display the results
print(f"Accuracy: {accuracy_ann:.2f}")
print("Classification Report:\n", classification_report_result_ann)
# Import necessary library
import matplotlib.pyplot as plt

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr_ann, tpr_ann, color='blue', label=f'ROC curve (area = {roc_auc_ann:.2f})')
plt.plot([0, 1], [0, 1], color='red', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```

 # Hyper Parameter Tuning for ANN On the Dataset Consisting Most Important Attributes

```
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Input, Dense, Dropout
from tensorflow.keras.models import Sequential
import matplotlib.pyplot as plt
# Define the number of features
num_features = X2_encoded_train.shape[1]
model3= Sequential([
    Input(shape=(num_features,)),
    Dense(64, activation='relu'),
    Dropout(0.2),
    Dense(32, activation='relu'),
    Dropout(0.2),
    Dense(1, activation='sigmoid')
])

model3.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

```
history = model3.fit(X2_scaled_train, Y2_encoded_train, epochs=10, batch_size=64,
validation_split=0.2)

test_loss, test_accuracy = model3.evaluate(X2_scaled_test, Y2_encoded_test, verbose=0)

print("Test Accuracy:", test_accuracy)

plt.plot(history.history['val_accuracy'], label='Test Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Accuracy')
plt.legend()
plt.show()
```

 # Logistic Regression for Lung Cancer Prediction

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, roc_curve,
auc
import matplotlib.pyplot as plt
X1_scaled_test, Y1_encoded_train, Y1_encoded_test
model = LogisticRegression()
model.fit(X1_scaled_train, Y1_encoded_train)
Y_pred = model.predict(X1_scaled_test)
accuracy = accuracy_score(Y1_encoded_test, Y_pred)
classification_report_result = classification_report(Y1_encoded_test, Y_pred)
confusion_matrix_result = confusion_matrix(Y1_encoded_test, Y_pred)
print(f"Accuracy: {accuracy:.2f}")
print("Classification Report:\n", classification_report_result)
print("Confusion Matrix:\n", confusion_matrix_result)
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

# Assuming Y_pred contains the predictions from your logistic regression model
# Generate predictions (Y_pred should already be calculated in your previous steps)
Y_pred = model.predict(X1_scaled_test)

# Compute the confusion matrix
cm = confusion_matrix(Y1_encoded_test, Y_pred)

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
ax = sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False,
                 xticklabels=['Negative', 'Positive'],  # Change the order if needed
                 yticklabels=['Negative', 'Positive'])

# Set the x-axis labels and move them to the top
ax.set_xticklabels(['Negative', 'Positive'])
ax.xaxis.set_ticks_position('top')
```

```
plt.xticks(rotation=0)
plt.ylabel('Actual Values')
plt.title('Confusion Matrix Heatmap', y=1.2, fontsize=16)
plt.figtext(0.5, 0.95, 'Predicted Values', ha='center', fontsize=12)

# Add text labels for true/false positives/negatives
labels = np.array([['True Negative', 'False Positive'], ['False Negative', 'True Positive']])
for i in range(2):
    for j in range(2):
        ax.text(j + 0.5, i + 0.3, f"{labels[i, j]}", ha='center', va='center', color='black', fontsize=10)

plt.show()
```

#visualization of Feature Importance for Logistic Regression

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.inspection import permutation_importance
from sklearn.linear_model import LogisticRegression

# Assuming you have a list of feature names
feature_names = ['age', 'smoking', 'chronic disease', 'fatigue', 'allergy', 'wheezing',
                 'alcohol consuming', 'coughing', 'shortness of breath',
                 'swallowing difficulty', 'chest pain']

# Create an instance of the Logistic Regression model (assuming it's already trained)
model = LogisticRegression()
model.fit(X1_scaled_train, Y1_encoded_train)

perm_importance = permutation_importance(model, X1_scaled_test, Y1_encoded_test,
n_repeats=10, random_state=42, scoring='accuracy')
feature_importances = np.abs(perm_importance.importances_mean)  # Take absolute values
feature_importances_normalized = (feature_importances - feature_importances.min()) /
(feature_importances.max() - feature_importances.min())
for name, importance in zip(feature_names, feature_importances_normalized):
    print(f"{name}: {importance:.4f}")
sorted_idx = np.argsort(feature_importances_normalized)[-15:]
plt.figure(figsize=(10, 8))
plt.barh(range(len(sorted_idx)), feature_importances_normalized[sorted_idx])
plt.yticks(range(len(sorted_idx)), np.array(feature_names)[sorted_idx])
plt.title("Feature Importances for Logistic Regression")
plt.ylabel("Feature")
plt.show()

Y3=Y1
print(Y3)
```

#Discarding features with Negligible Feature Importance

```
print("Original DataFrame:")
print(X1.head())
columns_to_drop = ['chronic disease']
columns_to_drop = [col for col in columns_to_drop if col in X1.columns]
```

```
X3 = X1.drop(columns=columns_to_drop)#saving the the datset without low feature column to new datset
print("DataFrame after dropping specified columns:")
print(X3.head())
```

#Transformation of New Data with Most Important feature to Validate Logistic Regression

```
# Split the data into training and testing sets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.utils import resample
Y3=Y1# In order to save the original datset saving the dependedent set to Y2
X3_train, X3_test, Y3_train, Y3_test = train_test_split(X3, Y3, test_size=0.2, random_state=42)
label_encoder = LabelEncoder()
Y3_encoded_train = label_encoder.fit_transform(Y3_train)
Y3_encoded_test = label_encoder.transform(Y3_test)
X3_encoded_train=X3_train# we dont nee one hot encoding as it is already in encoded from
X3_encoded_test=X3_test
scaler = StandardScaler()
X3_scaled_train = scaler.fit_transform(X3_encoded_train)
X3_scaled_test = scaler.transform(X3_encoded_test)

print(X3_encoded_train)
```

#Evaluating Performance of Logistic Regression on the Dataet with only Important Features

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, roc_curve, auc
import matplotlib.pyplot as plt
X1_scaled_test, Y1_encoded_train, Y1_encoded_test
model = LogisticRegression()
model.fit(X3_scaled_train, Y3_encoded_train)
Y_pred = model.predict(X3_scaled_test)
accuracy = accuracy_score(Y3_encoded_test, Y_pred)
classification_report_result = classification_report(Y3_encoded_test, Y_pred)
confusion_matrix_result = confusion_matrix(Y3_encoded_test, Y_pred)
print(f"Accuracy: {accuracy:.2f}")
print("Classification Report:\n", classification_report_result)
print("Confusion Matrix:\n", confusion_matrix_result)
plt.figure()
plt.plot([0, 1], [0, 1], color='grey', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc="lower right")
plt.show()
```

 # Calculating Probability of Having Lung Cancer for Individual patient Using LogisticRegression

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, roc_curve, auc
import matplotlib.pyplot as plt

# Assuming you have already split your data into training and test sets: X3_scaled_train, X3_scaled_test, Y3_encoded_train, Y3_encoded_test

# Create an instance of the Logistic Regression model
model = LogisticRegression()

# Train the model on the training data
model.fit(X3_scaled_train, Y3_encoded_train)

# Make predictions on the test set
Y_pred = model.predict(X3_scaled_test)

# Calculate the probabilities for each patient
probabilities = model.predict_proba(X3_scaled_test)[:, 1]

# Display the probability for each patient
for i, probability in enumerate(probabilities):
    print(f"Patient {i+1} Probability of Having Lung Cancer: {probability:.4f}")

# Evaluate the model's performance
accuracy = accuracy_score(Y3_encoded_test, Y_pred)
classification_report_result = classification_report(Y3_encoded_test, Y_pred)
confusion_matrix_result = confusion_matrix(Y3_encoded_test, Y_pred)

# Display the results
print(f"\nAccuracy: {accuracy:.2f}")
print("Classification Report:\n", classification_report_result)
print("Confusion Matrix:\n", confusion_matrix_result)
```

 # probability of having lung cancer for the first three patients using logistic regression are #0.83(83%),0.8650(86.5%),0.9911(99.11%) refers prediction of #of lung cancer as Yes,Yes,Yes,No accurate to the actual value of Y1_test or #Y3_encoded_test(1=Yes,0=No)

```
Y1_test
Y3_encoded_test
```

#Lung Cancer PredicTion Using Gaussian Naive Bayes Classifier

```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import roc_curve, accuracy_score, classification_report, confusion_matrix, auc
import matplotlib.pyplot as plt
# Create an instance of the Naive Bayes model
```

```
model = GaussianNB()
# Train the model on the training data
model.fit(X1_scaled_train, Y1_encoded_train)
# Make predictions on the test set
Y_pred = model.predict(X1_scaled_test)
Y_prob = model.predict_proba(X1_scaled_test)[:, 1]  # Probability estimates for the positive
class
# Evaluate the model's performance
accuracy = accuracy_score(Y1_encoded_test, Y_pred)
classification_report_result = classification_report(Y1_encoded_test, Y_pred)
confusion_matrix_result = confusion_matrix(Y1_encoded_test, Y_pred)
fpr, tpr, thresholds = roc_curve(Y1_encoded_test, Y_prob)
roc_auc = auc(fpr, tpr)
# Display the results
print(f"Accuracy: {accuracy:.2f}")
print("Classification Report for Naive Bayes Classifier:\n", classification_report_result)
print("Confusion Matrix:\n", confusion_matrix_result)
print("ROC AUC:", roc_auc)
# Plot the ROC curve
plt.figure()
plt.plot(fpr, tpr, color='blue', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='grey', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```

#Confusion Matrix Heatmap for Naive Bayes Classifier

```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
# Assuming Y_pred contains the predictions from your logistic regression model
# Generate predictions (Y_pred should already be calculated in your previous steps)
Y_pred = model.predict(X1_scaled_test)
# Compute the confusion matrix
cm = confusion_matrix(Y1_encoded_test, Y_pred)
# Plot the confusion matrix
plt.figure(figsize=(8, 6))
ax = sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False,
                 xticklabels=['Negative', 'Positive'],  # Change the order if needed
                 yticklabels=['Negative', 'Positive'])
print("confusion Matrix for Naive Bayes Classifier")
# Set the x-axis labels and move them to the top
ax.set_xticklabels(['Negative', 'Positive'])
ax.xaxis.set_ticks_position('top')
plt.xticks(rotation=0)
```

```
plt.ylabel('Actual Values')
plt.title('Confusion Matrix Heatmap', y=1.2, fontsize=16)
plt.figtext(0.5, 0.95, 'Predicted Values', ha='center', fontsize=12)
# Add text labels for true/false positives/negatives
labels = np.array([['True Negative', 'False Positive'], ['False Negative', 'True Positive']])
for i in range(2):
    for j in range(2):
        ax.text(j + 0.5, i + 0.3, f"{labels[i, j]}", ha='center', va='center', color='black', fontsize=10)

plt.show()
```

 # Visualizing Feature Importance for Gaussian Naive Bayes Classifier

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, cross_val_score, KFold
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, roc_curve,
auc
from sklearn.inspection import permutation_importance
import matplotlib.pyplot as plt
# Assuming you have already defined X and y, and performed train-test split, encoding, and
scaling
# Instantiate the Naive Bayes model
model = GaussianNB()
# Train the model on the entire training data
model.fit(X1_scaled_train, Y1_encoded_train)
# Calculate permutation feature importance
perm_importance = permutation_importance(model, X1_scaled_test, Y1_encoded_test,
n_repeats=10, random_state=42, scoring='accuracy')
# Get the feature importances
feature_importances = np.abs(perm_importance.importances_mean)  # Take absolute values
# Normalize the feature importances to be within the range of 0 to 1
feature_importances_normalized = (feature_importances - feature_importances.min()) /
(feature_importances.max() - feature_importances.min())
# Assuming you have a list of feature names
feature_names = X1_encoded_train.columns
# Print the normalized feature importances with their names
for name, importance in zip(feature_names, feature_importances_normalized):
    print(f"{name}: {importance:.4f}")
# Sort the feature importances in descending order
sorted_idx = np.argsort(feature_importances_normalized)[-11:]
# Plot the normalized feature importances
plt.figure(figsize=(10, 8))
plt.barh(range(len(sorted_idx)), feature_importances_normalized[sorted_idx])
plt.yticks(range(len(sorted_idx)), np.array(feature_names)[sorted_idx])
plt.title(" Feature Importances for Gsussian Naive Bayes Classifier")
plt.show()
```

 # Discarding Features Having Negligible Feature Importance

```
import pandas as pd
# Columns to be deleted
columns_to_drop = ['allergy','coughing']
# Ensure the columns to drop are present in the DataFrame
columns_to_drop = [col for col in columns_to_drop if col in X1.columns]
# Drop the specified columns
X4= X1.drop(columns=columns_to_drop)#saving the the datset without low feature column to
new datset
print("DataFrame after dropping specified columns for Naive bayes classifier:")
print(X4.head())
```

#Transformation of Dataset for using in Gaussian Naive Bayes Classifier

```
# Split the data into training and testing sets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.utils import resample
Y4=Y1# In order to save the original datset saving the dependedent set to Y2
# Split the data into training and testing sets
X4_train, X4_test, Y4_train, Y4_test = train_test_split(X4, Y4, test_size=0.2, random_state=42)
# Assuming 'target_column' is the target variable in Y
label_encoder = LabelEncoder()
Y4_encoded_train = label_encoder.fit_transform(Y4_train)
Y4_encoded_test = label_encoder.transform(Y4_test)
# Standardize the features
X4_encoded_train=X4_train# we dont nee one hot encoding as it is already in encoded from
X4_encoded_test=X4_test
scaler = StandardScaler()
X4_scaled_train = scaler.fit_transform(X4_encoded_train)
X4_scaled_test = scaler.transform(X4_encoded_test)
print(X4_encoded_train)
```

 # Evaluating performance of Gaussian Naive Bayes Classifier on the Dataset Having only #Important Attributes

```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import roc_curve, accuracy_score, classification_report, confusion_matrix,
auc
import matplotlib.pyplot as plt
# Create an instance of the Naive Bayes model
model = GaussianNB()
# Train the model on the training data
model.fit(X4_scaled_train, Y4_encoded_train)
# Make predictions on the test set
Y_pred = model.predict(X4_scaled_test)
# Evaluate the model's performance
accuracy = accuracy_score(Y4_encoded_test, Y_pred)
classification_report_result = classification_report(Y4_encoded_test, Y_pred)
confusion_matrix_result = confusion_matrix(Y4_encoded_test, Y_pred)
# Display the results
```

```
print(f"Accuracy: {accuracy:.2f}")
print("Classification Report:\n", classification_report_result)
print("Confusion Matrix:\n", confusion_matrix_result)
```

 # Calculating Probability of Having Lung Cancer for Individual Patient using Gaussian #Naive Bayes Classifier

```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import roc_curve, accuracy_score, classification_report, confusion_matrix,
auc
import matplotlib.pyplot as plt
# Create an instance of the Naive Bayes model
model = GaussianNB()
# Train the model on the training data
model.fit(X4_scaled_train, Y4_encoded_train)
# Make predictions on the test set
Y_pred = model.predict(X4_scaled_test)
# Calculate the probabilities of each class
Y4_prob = model.predict_proba(X4_scaled_test)
# Display the probabilities for lung cancer (assuming class 1 is lung cancer)
lung_cancer_probabilities = Y4_prob[:, 1]
# Print the probabilities for each patient
print("Probabilities of Lung Cancer for Each Patient:")
for i, prob in enumerate(lung_cancer_probabilities):
    print(f"Patient {i + 1}: {prob:.2f}")
Y1_test
Y4_encoded_test
```

#Calculating the probability of having lung cancer for each patients using Gaussian Naive Bayes #Classifier the first four patients outcome is 1.00(100%), #1.00(100%),1(100%),0(0) which is same as the actual outcome lung cancer in test dataset #Y_test or Y_encoded_test,i.e,Yes,Yes,Yes,No

#Hyperparameter Tuning for Gaussian Naive Bayes Classifier And calculating Lung #Cancer probability using the tuned model

```
import numpy as np
from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import roc_curve, accuracy_score, classification_report, confusion_matrix,
auc
import matplotlib.pyplot as plt
# Define the Gaussian Naive Bayes model
model = GaussianNB()
# Define the parameter grid for var_smoothing
param_grid = {'var_smoothing': np.logspace(0, -9, num=100)}
# Create the GridSearchCV object
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5,
scoring='accuracy')
# Fit the model on the training data
grid_search.fit(X4_scaled_train, Y4_encoded_train)
```

```
# Get the best model with the tuned hyperparameter
best_model = grid_search.best_estimator_

# Make predictions on the test set using the best model
Y_pred = best_model.predict(X4_scaled_test)

# Calculate the probabilities of each class using the best model
Y4_prob = best_model.predict_proba(X4_scaled_test)

# Display the probabilities for lung cancer (assuming class 1 is lung cancer)
lung_cancer_probabilities = Y4_prob[:, 1]

# Print the probabilities for each patient
print("Probabilities of Lung Cancer for Each Patient after Hyperparameter Tuning:")
for i, prob in enumerate(lung_cancer_probabilities):
    print(f"Patient {i + 1}: {prob:.4f}")
# Evaluate the model's performance
accuracy = accuracy_score(Y4_encoded_test, Y_pred)
classification_report_result = classification_report(Y4_encoded_test, Y_pred)
confusion_matrix_result = confusion_matrix(Y4_encoded_test, Y_pred)
# Compute ROC curve and ROC AUC using the best model
fpr, tpr, thresholds = roc_curve(Y4_encoded_test, lung_cancer_probabilities)
roc_auc = auc(fpr, tpr)
# Display the results
print(f"\nBest var_smoothing: {grid_search.best_params_['var_smoothing']}")
print("Classification Report after hyperparameter tuning:\n", classification_report_result)
print("Confusion Matrix after hyperparameter tuning:\n", confusion_matrix_result)
print("ROC AUC after hyperparameter tuning:", roc_auc)
# Plot ROC curve
plt.figure()
plt.plot(fpr, tpr, color='blue', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='grey', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve after Hyperparameter Tuning')
plt.legend(loc="lower right")
plt.show()
best_cv_accuracy = grid_search.best_score_
print(f" Accuracy after hyperparameter tuning: {best_cv_accuracy:.2f}")
# Optional: Show CV accuracy for all parameter settings tested
cv_results = grid_search.cv_results_
# Extract mean cross-validated accuracy for each parameter setting
mean_test_scores = cv_results['mean_test_score']
# Print the top 5 parameter settings based on CV accuracy
top_indices = np.argsort(mean_test_scores)[-5:][::-1]
for idx in top_indices:
    print(f"var_smoothing: {cv_results['param_var_smoothing'].data[idx]}, CV Accuracy:
{mean_test_scores[idx]:.4f}")
```

#Lung Cancer Detection using SVM

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, roc_curve,
auc
import matplotlib.pyplot as plt

# Assuming X_scaled_train, X_scaled_test, Y_encoded_train, Y_encoded_test are defined
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, roc_curve,
auc
# Define and train the SVM classifier model with additional hyperparameters
model = SVC(
    kernel='linear',
    C=1,
    probability=True,
    random_state=42,
    tol=1e-4,
    max_iter=-1,
    class_weight=None,
    verbose=False,
    shrinking=True,
    decision_function_shape='ovr',
    break_ties=False
)
# Fit the model on the training data
model.fit(X1_scaled_train, Y1_encoded_train)
# Make predictions on the test set
Y_pred = model.predict(X1_scaled_test)
# Decode the encoded target variable
label_encoder = LabelEncoder()
label_encoder.fit(Y1_encoded_test)
Y_test_decoded = label_encoder.inverse_transform(Y1_encoded_test)
Y_pred_decoded = label_encoder.inverse_transform(Y_pred)
# Evaluate the model's performance
accuracy = accuracy_score(Y_test_decoded, Y_pred_decoded)
classification_report_result = classification_report(Y_test_decoded, Y_pred_decoded)
confusion_matrix_result = confusion_matrix(Y_test_decoded, Y_pred_decoded)
# Display the results
print("Accuracy:", accuracy)
print("Classification Report:\n", classification_report_result)
print("Confusion Matrix:\n", confusion_matrix_result)
# Plot ROC curve
```

#Confusion Matrix heatmap for SVM

```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
Y_pred = model.predict(X1_scaled_test)
# Compute the confusion matrix
cm = confusion_matrix(Y1_encoded_test, Y_pred)
# Plot the confusion matrix
plt.figure(figsize=(8, 6))
ax = sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False,
                 xticklabels=['Negative', 'Positive'],
                 yticklabels=['Negative', 'Positive'])
print("confusion Matrix for SVM Bayes Classifier")
# Set the x-axis labels and move them to the top
ax.set_xticklabels(['Negative', 'Positive'])
ax.xaxis.set_ticks_position('top')
plt.xticks(rotation=0)
plt.ylabel('Actual Values')
plt.title('Confusion Matrix Heatmap', y=1.2, fontsize=16)
plt.figtext(0.5, 0.95, 'Predicted Values', ha='center', fontsize=12)
# Add text labels for true/false positives/negatives
labels = np.array([['True Negative', 'False Positive'], ['False Negative', 'True Positive']])
for i in range(2):
    for j in range(2):
        ax.text(j + 0.5, i + 0.3, f"{labels[i, j]}", ha='center', va='center', color='black', fontsize=10)

plt.show()
```

#Feature Importance for SVM

```
from sklearn.model_selection import train_test_split, cross_val_score, KFold
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, roc_curve,
auc
from sklearn.inspection import permutation_importance
import matplotlib.pyplot as plt

# Assuming you have already defined X and y, and performed train-test split, encoding, and
#scaling

model = SVC(
    kernel='linear',
    C=1,
    probability=True,
    random_state=42,
    tol=1e-4,
    max_iter=-1,
    class_weight=None,
    verbose=False,
    shrinking=True,
    decision_function_shape='ovr',
```

```
    break_ties=False
)

# Train the model on the entire training data
model.fit(X1_scaled_train, Y1_encoded_train)
# Calculate permutation feature importance
perm_importance = permutation_importance(model, X1_scaled_test, Y1_encoded_test,
n_repeats=10, random_state=42, scoring='accuracy')
# Get the feature importances
feature_importances = np.abs(perm_importance.importances_mean)  # Take absolute values
# Normalize the feature importances to be within the range of 0 to 1
feature_importances_normalized = (feature_importances - feature_importances.min()) /
(feature_importances.max() - feature_importances.min())
# Assuming you have a list of feature names
feature_names = X1_encoded_train.columns
# Print the normalized feature importances with their names
for name, importance in zip(feature_names, feature_importances_normalized):
    print(f"{name}: {importance:.4f}")
# Sort the feature importances in descending order
sorted_idx = np.argsort(feature_importances_normalized)[-11:]
# Plot the normalized feature importances
plt.figure(figsize=(10, 8))
plt.barh(range(len(sorted_idx)), feature_importances_normalized[sorted_idx])
plt.yticks(range(len(sorted_idx)), np.array(feature_names)[sorted_idx])
plt.title(" Feature Importances for SVM ")
plt.show()
```

#Discarding Features with Neglizible(0) importance

```
import pandas as pd
# Columns to be deleted
columns_to_drop = ['chest pain','coughing']
# Ensure the columns to drop are present in the DataFrame
columns_to_drop = [col for col in columns_to_drop if col in X1.columns]
# Drop the specified columns
X5= X1.drop(columns=columns_to_drop)#saving the the datset without low feature column to
new datset
print("DataFrame after dropping specified columns for SVM classifier:")
print(X5.head())
# Split the data into training and testing sets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.utils import resample
Y5=Y1# In order to save the original datset saving the dependedent set to Y2

# Split the data into training and testing sets
X5_train, X5_test, Y5_train, Y5_test = train_test_split(X5, Y5, test_size=0.2, random_state=42)
# Assuming 'target_column' is the target variable in Y
label_encoder = LabelEncoder()
Y5_encoded_train = label_encoder.fit_transform(Y5_train)
Y5_encoded_test = label_encoder.transform(Y5_test)
```

```
# Standardize the features
X5_encoded_train=X5_train# we dont nee one hot encoding as it is already in encoded from
X5_encoded_test=X5_test
scaler = StandardScaler()
X5_scaled_train = scaler.fit_transform(X5_encoded_train)
X5_scaled_test = scaler.transform(X5_encoded_test)

print(X5_encoded_train)
```

 # Evaluating performance of the SVM Classifier on the datset having only important features

```
model =  SVC(
    kernel='linear',
    C=1,
    probability=True,
    random_state=42,
    tol=1e-4,
    max_iter=-1,
    class_weight=None,
    verbose=False,
    shrinking=True,
    decision_function_shape='ovr',
    break_ties=False
)


# Train the model on the training data
model.fit(X5_scaled_train, Y5_encoded_train)

# Make predictions on the test set
Y_pred = model.predict(X5_scaled_test)

# Evaluate the model's performance
accuracy = accuracy_score(Y5_encoded_test, Y_pred)
classification_report_result = classification_report(Y5_encoded_test, Y_pred)
confusion_matrix_result = confusion_matrix(Y5_encoded_test, Y_pred)
fpr, tpr, thresholds = roc_curve(Y5_encoded_test, Y_pred, pos_label=model.classes_[1])
roc_auc = auc(fpr, tpr)

# Display the results
print(f"Accuracy: {accuracy:.2f}")
print("Classification Report:\n", classification_report_result)
print("Confusion Matrix:\n", confusion_matrix_result)
print("ROC AUC:", roc_auc)

# Plot ROC Curve
plt.figure()
plt.plot(fpr, tpr, color='blue', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='grey', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
```

```
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```

#Calculating probability of Lung Cancer for Individual Patient Using SVM on dataset having most #important features

```
model =  SVC(
    kernel='linear',
    C=1,
    probability=True,
    random_state=42,
    tol=1e-4,
    max_iter=-1,
    class_weight=None,
    verbose=False,
    shrinking=True,
    decision_function_shape='ovr',
    break_ties=False
)

# Train the model on the training data
model.fit(X5_scaled_train, Y5_encoded_train)

# Make predictions on the test set
Y_pred = model.predict(X5_scaled_test)

# Calculate the probabilities of each class
Y4_prob = model.predict_proba(X5_scaled_test)

# Display the probabilities for lung cancer (assuming class 1 is lung cancer)
lung_cancer_probabilities = Y4_prob[:, 1]

# Print the probabilities for each patient
print("Probabilities of Lung Cancer for Each Patient:")
for i, prob in enumerate(lung_cancer_probabilities):
    print(f"Patient {i + 1}: {prob:.2f}")

#Comparing result of having lung cancer with the test set
Y1_test
Y5_encoded_test
```

#Comparison of Classification report of all the propesed model

```
# Example metrics
models = ['ANN', 'Logistic Regression', 'Naive Bayes Classifier','SVM']
accuracies = [ 0.96, 0.97, 0.94, 0.97]
precisions = [0.94, 0.97, 0.94,0.97]
recalls = [0.94 , 0.97,   0.94, 0.97 ]
f1_scores = [0.94 , 0.97,   0.94,0.97 ]
```

```
x = range(len(models))  # Position of bars on x-axis
# Define the data
models = ['ANN', 'Logistic Regression', 'Naive Bayes Classifier', 'SVM']
accuracies = [0.96, 0.97, 0.94, 0.97]
precisions = [0.94, 0.97, 0.94, 0.97]
recalls = [0.94, 0.97, 0.94, 0.97]
f1_scores = [0.94, 0.97, 0.94, 0.97]

# Create a DataFrame
df = pd.DataFrame({
    'Model': models,
    'Accuracy': accuracies,
    'Precision': precisions,
    'Recall': recalls,
    'F1 Score': f1_scores
})

# Print the DataFrame

print(df)



print(" classification report for all proposed models")



#Comparitive analysis of all models accuracy using chart plot
plt.figure(figsize=(18, 6))

# Accuracy Bar Chart
plt.subplot(2, 2, 1)
plt.bar(x, accuracies, color='b', alpha=0.7)
plt.xticks(x, models)
plt.ylabel('Accuracy')
plt.title('Model Accuracy')


plt.tight_layout()
plt.show()

#Comparitive analysis of precision of all proposed model using chart plot
plt.figure(figsize=(18, 6))
plt.subplot(2, 2, 2)
plt.bar(x, precisions, color='g', alpha=0.7)
plt.xticks(x, models)
plt.ylabel('Precision')
plt.title('Model Precision')
plt.tight_layout()
plt.show()
```

```
plt.figure(figsize=(18, 6))

#Comparitive analysis of Recall  of all proposed model using chart plot
plt.subplot(2, 2, 3)
plt.bar(x, recalls, color='r', alpha=0.7)
plt.xticks(x, models)
plt.ylabel('Recall')
plt.title('Model Recall')


#Comparitive analysis of F1 score of all proposed model  using chart plot
plt.figure(figsize=(18, 6))
plt.subplot(2, 2, 4)
plt.bar(x, f1_scores, color='purple', alpha=0.7)
plt.xticks(x, models)
plt.ylabel('F1 Score')
plt.title('Model F1 Score')
```