ABSTRACT

Metastasis, the spread of cancer cells from the primary tumor to distant organs or tissues, is a crucial aspect of cancer progression that significantly affects patient outcomes. Detecting the approximate survival time for patients with metastatic cancer is of paramount importance for tailoring treatment plans and improving patient care.Existing studies have shown promising results in predicting survival times for certain cancer types, but the complexity and heterogeneity of metastatic cancer necessitate further advancements. Despite advancements, there exists a gap in predicting whether lung cancer will spread to other parts, estimating patients' survival time, and identifying key factors influencing metastasis. Existing research has laid the foundation by addressing some aspects, but a comprehensive approach integrating these components is lacking. In response, this proposal aims to leverage the power of Ensemble Learning to bridge this gap and achieve improved accuracy, robustness, and generalization in predicting lung cancer metastasis. Existing models often overlook the unique challenges posed by metastatic cancer, such as the dynamic nature of tumor progression and the influence of the tumor microenvironment on metastatic behavior and do not usually use textual data for detection.

Various Ensemble models are trained on data obtained from SEER. SEER provides validated cancer data of anonymous patients for which access permit is required. Upon accessing the database,necessary data is filtered and fetched. Exploratory data analysis is performed on data to extract and visualize trends in data. Data preprocessing steps are implemented and models are trained. Following the testing and validation of performance of models using relevant plots the optimal model is chosen. The chosen model is deployed onto the website designed to make real time predictions based on user details.

For metastasis detection, the random forest model was the best model with accuracy of 93.66%, recall of 0.99 and precision of 0.94. For the approximate survival time prediction which is a multiclass classification problem the gradient boosting classifier was the best model obtained with mean error of 0.69 months, training accuracy of 72% and testing accuracy of 61%. There is no existing implementation of the above 2 types of prediction being made on the same dataset that is used.

Having tested the different aspects of the software application and analysing the data from the SEER registry,this application can definitely be refined with a more polished UI as well as training it with more data will most certainly result in a viable product. The product provides a novel approach and high accuracy in detecting both Survival Time and Metastasis with the Random Forest model and the important features detected from the model can be further analyzed by medical practitioners to see any resulting correlations.All the machine learning models showcased above were trained only on a subset of the data that was received from the National Cancer Institute. Although our application is focused only on Lung Cancer, it can be extended to predict Metastasis and Survival Time for other types of Cancer too. It will be interesting to see the patterns observed if Unsupervised Learning techniques are applied.

CHAPTER 1

Introduction

1.1 State of Art Development

Researchers explore various ensemble algorithms, including deep learning techniques, to improve predictive accuracy and handle high-dimensional data like genomics and medical images. Addressing class imbalance and utilizing survival analysis methods further enhance model performance. This cutting-edge research emphasizes model interpretability and explainability, facilitating collaboration between machine learning and medical experts. As an ever-evolving field, staying up-to-date with the latest research is crucial to advance cancer prognosis and personalized treatment strategies.

There are multiple different approaches to Cancer metastasis prediction employed in the status quo. Given below are some of the techniques used:

1. Deep Learning and Convolutional Neural Networks (CNNs)
2. Multi-Omics Data Integration
3. Integration of Clinical and Molecular Data
4. Predictive Biomarkers
5. Real-time Monitoring and Predictive Analytics

1.2 Motivation

The motivation for research on metastasis detection and survival time prediction using ensemble learning stems from the urgent need to improve cancer patient outcomes. Metastasis is a critical stage of cancer progression, and accurate predictions of survival times can significantly impact treatment decisions and patient care. Traditional prognostic approaches have limitations in handling the complexity and heterogeneity of metastatic tumors. Ensemble learning techniques offer the potential to harness the collective power of diverse models and data sources, promising more accurate, robust, and clinically applicable predictions. Advancements in this field could lead to personalized treatment strategies, better resource allocation, and ultimately contribute to enhancing the overall quality of life for cancer patients. The project aims to provide medical practitioners with a tool that identifies potential metastatic tendencies early, enabling them to take proactive steps to manage the disease and tailor treatment plans to individual patients.

1.3 Problem Statement

The problem at hand is to develop an accurate and clinically applicable metastasis detection and survival time prediction model using ensemble learning techniques. The objective is to leverage diverse datasets, including clinical, genomic, and imaging data, to improve the accuracy and

robustness of predictions for cancer patients with metastatic tumors. The research aims to address the challenges posed by the dynamic nature of tumor progression and the influence of the tumor microenvironment on metastatic behavior. The ultimate goal is to create a reliable and interpretable tool that aids oncologists in estimating approximate survival times, facilitating personalized treatment planning and improving patient outcomes in the context of metastatic cancer.

1.4 Objectives

1. Identification of Common Evaluation Metric: The objective is to develop a comprehensive evaluation metric that not only assesses the technical performance of various ensemble architectures but also considers their practical utility in real-life clinical settings. This metric will bridge the gap between AI research and healthcare application by aligning model effectiveness with the needs and constraints of clinics and hospitals.

2. Ensemble Technique Investigation and Model Validation: This objective involves a thorough investigation and comparison of multiple ensemble techniques to determine the most effective approach for predicting lung cancer metastasis. The goal is to validate the selected model rigorously using real-world patient data, ensuring its reliability and accuracy for practical clinical use.

3. Enhancement of Prediction Accuracy and Efficiency: Building upon insights from literature, this objective focuses on enhancing the accuracy and efficiency of metastasis prediction methods. By refining parameters, curating high-quality datasets, and implementing optimization strategies, the project aims to push the boundaries of current predictive capabilities, ultimately leading to more accurate and timely predictions.

4. User-Friendly Interface and Deployment Pipeline: The project seeks to create a user-friendly interface and an end-to-end deployment pipeline tailored to the operational realities of hospitals. This involves considering constraints like computational resources and existing hospital software, ensuring seamless integration of the model into clinical workflows for easy adoption by medical professionals and staff.

5. Contribution to Research on Metastasis: This objective aims to contribute to the broader field of oncology by identifying specific attributes and underlying causes that drive the spread of tumor cells in lung cancer patients. By leveraging AI techniques, the project endeavors to provide novel insights into the complex mechanisms of metastasis, potentially guiding future research and treatment strategies.

1.5 Scope

The scope of the project involves developing an ensemble learning model using diverse datasets, including SEER data, for accurate metastasis detection and survival time prediction in cancer patients. The research will focus on feature engineering, algorithm selection, and model optimization to improve predictive accuracy. Validation and performance comparison will be conducted to ensure robustness. The model's clinical applicability will be emphasized, providing oncologists with an interpretable tool for personalized treatment planning.

1.6 Methodology

The methodology for this project involves the following steps:

- Data Collection: Gathering Dataset from SEER which is validated cancer data from NCI. The required data is filtered and fetched upon approval of access permit.
- Data Preprocessing: Applying image processing techniques to standardize data, remove noise, perform appropriate string to numerical mapping and normalization techniques.
- Model Development: Implementing various Ensemble Learning models and training the same on preprocessed Data.
- Graphical User Interface (GUI): Designing an intuitive GUI that allows users to input required medical details and obtain appropriate prediction
- Performance Evaluation: Assessing the system's performance using evaluation metrics, including the accuracy, precision, recall, and F1 score.
- Deployment: Deploying selected optimal model on GUI for real time processing and prediction.

CHAPTER 2

Literature Survey

2.1 Introduction

A literature survey on lung cancer metastasis prediction aims to explore and analyze the existing body of research focused on predicting the spread of lung cancer to distant organs or tissues. This survey seeks to identify the current state-of-the-art methods, challenges, and gaps in the field. By reviewing published papers, articles, and research studies, we aim to gain insights into the various approaches used for lung cancer metastasis prediction. This comprehensive review will provide a solid foundation for the development of a novel and effective predictive model that can aid in improving patient outcomes, treatment planning, and overall clinical decision-making for lung cancer patients.

2.2 Related Work

The literature survey on lung cancer metastasis prediction reveals several noteworthy studies and research works related to the topic. One study proposes a machine learning model that combines genetic biomarkers and clinical data to predict the likelihood of lung cancer metastasis with high accuracy. Another research effort focuses on using deep learning techniques, specifically convolutional neural networks (CNNs), to analyze lung cancer CT images and accurately classify cases with metastatic involvement. Additionally, the integration of radiomics and genomics data is explored to enhance lung cancer metastasis prediction. Another study employs Random Survival Forests, an ensemble learning method, to estimate the probability of lung cancer metastasis and the time to metastasis occurrence, effectively handling censored survival data. Moreover, the combination of multiple omics data, including genomics and proteomics, is studied to build a comprehensive prediction model for lung cancer metastasis. Comparative investigations into different ensemble learning methods for lung cancer metastasis prediction have been conducted. Furthermore, there are review articles that discuss the role of various clinical and pathological factors in predicting lung cancer metastasis, highlighting the significance of incorporating traditional prognostic factors into predictive models. These related works showcase the growing interest in using advanced machine learning techniques, integrating multimodal data, and exploring the role of imaging and genetic biomarkers to improve lung cancer metastasis prediction, providing valuable insights for the development of a novel and comprehensive predictive model in our research. A convolutional neural network (CNN) in predicting the potential of newly diagnosed non–small cell lung cancer (NSCLC) to metastasize to lymph nodes or distant sites. A systems biology approach for NSCLC patients, which identified eight novel survival-related genes based on seven previously well-known biomarkers. We integrated systems biology and deep learning approaches to predict the survival status of NSCLC patients. A comprehensive overview

of the recent advances in AI for lung cancer pathology image analysis. The paper highlights the potential benefits of using AI in these tasks, but it also discusses the challenges and opportunities in the use of AI. The performance of the ensemble classifier on the Breast Cancer Wisconsin Diagnostic (WDBC) dataset, and shows that it achieves an accuracy of 98.1%. The ensemble classifier achieves a high accuracy on the WDBC dataset, and it is relatively simple to implement.

2.3 Summary

The literature review on lung cancer metastasis prediction provides an overview of existing research in the field. Studies have utilized machine learning, deep learning, and ensemble methods to integrate diverse data sources, including genetic, clinical, and imaging data. Multi-omics data integration and radiomics-genomics fusion have been explored to improve predictive accuracy. The review also emphasizes the role of traditional prognostic factors in predictive models. The findings from related works inform the development of a comprehensive predictive model for enhanced lung cancer metastasis prediction.

CHAPTER 3

Software Requirements Specification

3.1 Functional Requirements

- **Main Page**

This page will explain the main objectives of the Lung Cancer Detection System.

- **Metastasis Detector**

This page will allow doctors to input details about a diagnosed patient's history and find out if there is a chance of the Cancer spreading.

- **Survival Time Detector**

This page will allow doctors to input details about a diagnosed patient's history and find out the approximate survival time for a patient diagnosed with Malignant Cancer.

- **Research Center**

This is the research done on SEER Research Data about the key factors that contribute to Metastasis. Apart from this, general analysis carried out on the Lung Cancer patients from the NCI will be displayed.

3.2 Non-Functional Requirements

- **Dependencies**

The application runs mainly on a system that has Python 3.5 on apache server.

In order to run some of the models Sklearn packages would be required.

- **Assumptions**

All aspects of the project will work in the way the designer assumed. All design flaws will be found early on. The project will meet the requirements and satisfy the customer. None of the hardware or software is stolen or sabotaged.

- **Performance**

The response time is the entire time beginning with the initial user action on the browser, the request going to the server, the response being received from the server, and finally it being processed by application. Production of a simple response shall take less than 2 seconds for 95% of the cases. Platform independent.

The system shall be able to handle up to 5000 concurrent users when satisfying all their requirements. The system shall predict with high accuracy if a patient is susceptible to Lung Cancer. The system shall redirect potential patients to well-reputed medical practitioners depending on the location.

- **Reliability**

The system is reliable 95% of the time, that is it produces the same output for a given input every time it is faced with the same input under similar working conditions.

- **Availability**

On an average the model can process the input and produce results 98% of the time.

- **Maintainability**

Maintaining the system involves minimal effort. Since there is no external hardware as such, maintenance of the entire system is very minimal.Only the models which are made for the different techniques will be altered from time to time.

3.3 Software Requirements

- Programming Language: Python 3
- Integrated Development Environment (IDE): PyCharm or Google Colab.
- Machine Learning Libraries: Utilize machine learning libraries to implement and train your models.

a. Scikit-learn

b. TensorFlow

- Data Manipulation and Visualization:

a. Pandas

b. NumPy

c. Matplotlib

3.4 Hardware Requirements

1. Processor (CPU): A powerful multi-core processor, such as Intel Core i7 or higher, or an equivalent processor from AMD.

2. Random Access Memory (RAM): Minimum 16 GB of RAM, preferably 32 GB or more, to handle large datasets and model training efficiently.

3. Graphics Processing Unit (GPU) (optional): NVIDIA GeForce RTX series or equivalent GPU, especially beneficial for accelerating the training of deep learning models.

4. Storage: Solid-state drive (SSD) with at least 50 GB of capacity to store datasets, trained models, and intermediate results.

3.4 Summary

The provided document outlines the functional and non-functional requirements for a Lung Cancer Detection System, detailing its main components and specifications. The system aims to assist doctors in diagnosing and predicting lung cancer, as well as providing research insights.
This section provides a comprehensive overview of the Lung Cancer Detection System's functional and non-functional requirements, as well as the necessary software and hardware components to develop and operate the system effectively.

CHAPTER 4

Design

4.1 High-Level Design

DFD is the abbreviation for Data Flow Diagram. The flow of data of a system or a process is represented by DFD. It also gives insight into the inputs and outputs of each entity and the process itself. DFD does not have control flow and no loops or decision rules are present. Specific operations depending on the type of data can be explained by a flowchart.
It is a graphical tool, useful for communicating with users ,managers and other personnel. It is useful for analyzing existing as well as proposed systems. It provides an overview of the various system processes, transformations that are performed, location where the data is stored and the results that are produced.

DFD Level 0

A DFD Level 0, also known as a context diagram, is the highest level of a data flow diagram (DFD). It provides an overview of the entire system and shows the major processes, data flows, and data stores in the system. A DFD Level 0 does not show any detail about the internal workings of the processes

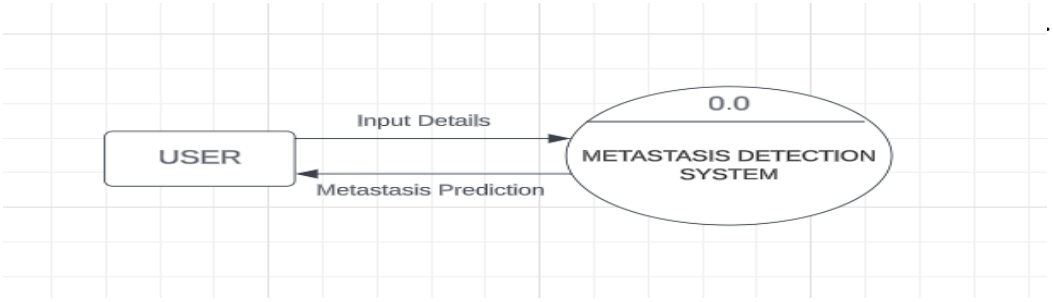

Fig 4.1 DFD level 0 for Lung Cancer Metastasis Detection

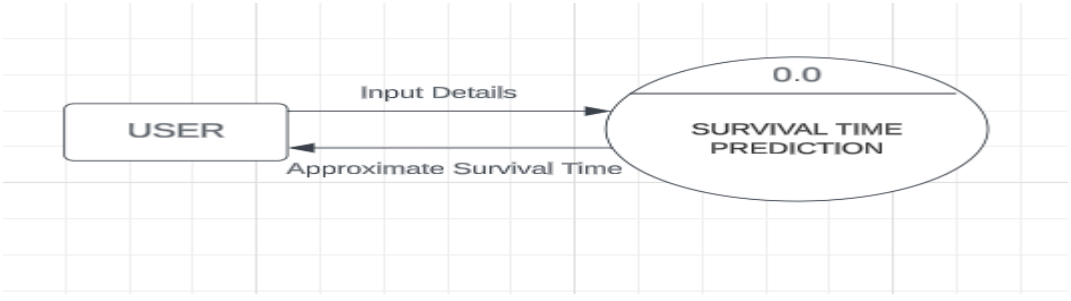

Fig 4.2 DFD level 0 for Survival Time Prediction

DFD Level 1

A DFD Level 1 is a data flow diagram that shows the major processes of a system in more detail than a DFD Level 0. It breaks down the major processes in the context diagram into smaller, more manageable processes. A DFD Level 1 is still a high-level view of the system, but it provides more detail than a DFD Level 0. It can be used to understand the overall architecture of the system and to identify the major processes that need to be implemented.

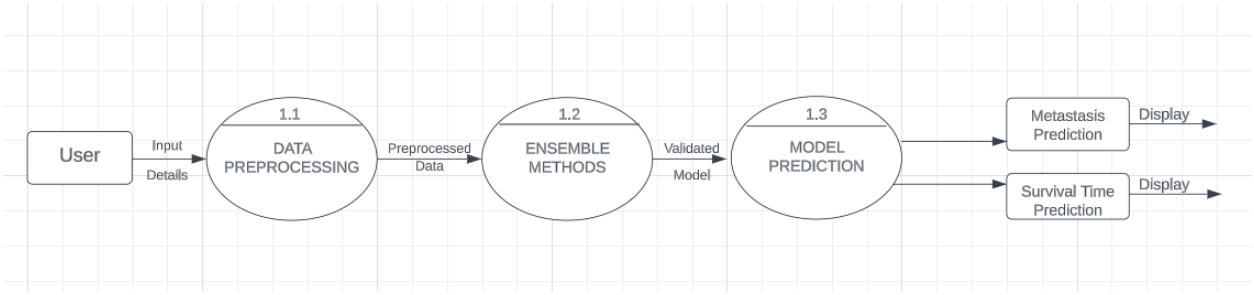

Fig 4.3 DFD level 1 for Lung Cancer Metastasis Detection

DFD Level 2

A DFD Level 2 is a data flow diagram that shows the internal workings of a process in more detail than a DFD Level 1. It breaks down the process into smaller, more manageable subprocesses. A DFD Level 2 is still a high-level view of the process, but it provides more detail than a DFD Level 1. It can be used to understand the detailed logic of the process and to identify the subprocesses that need to be implemented.

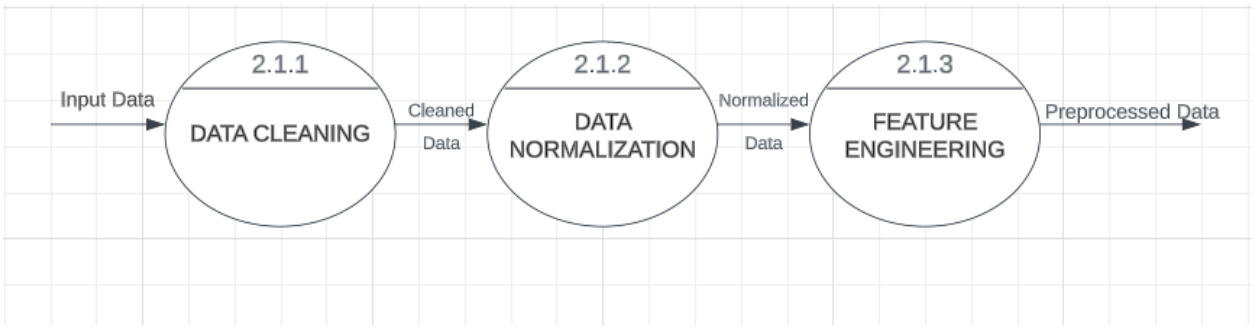

Fig 4.4 DFD level 2 for Data Preprocessing Sub-Module

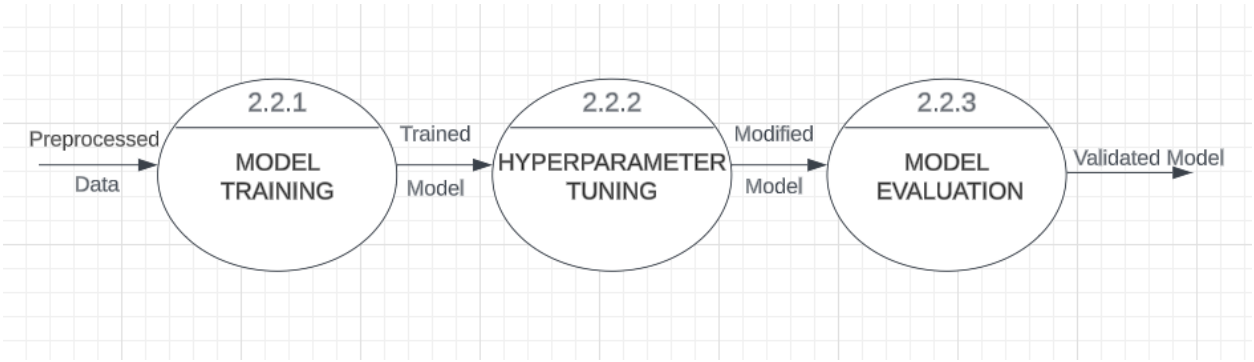

Fig 4.5 DFD level 2 for Ensemble Methods Sub-Module

4.1.1 System Architecture

- **Data Collection and Preprocessing Module**: This module is responsible for gathering diverse datasets, including clinical records, genomic data, and medical imaging, related to lung cancer patients. The collected data is preprocessed to handle missing values, normalize features, and address class imbalance.
- **Feature Engineering and Selection Module**: This component focuses on extracting relevant features from the preprocessed data. Techniques like radiomics analysis and genetic feature selection are applied to identify discriminative features that capture the complex characteristics of lung cancer metastasis.
- **Ensemble Learning Model Development Module**: Multiple machine learning algorithms are employed to create an ensemble learning model. Algorithms such as Random Forests, Gradient Boosting, and Deep Neural Networks are combined to leverage their collective predictive power. Survival Analysis Module: This module is responsible for integrating survival analysis techniques into the ensemble learning model. Models like Random Survival Forests or survival loss functions in gradient boosting algorithms handle censored survival data and predict survival times.
- **Model Validation and Performance Evaluation Module**: The developed ensemble learning model is validated using cross-validation techniques to assess its robustness and generalization performance. Evaluation metrics like accuracy, sensitivity, specificity, and concordance index are used to measure model performance.
- **Clinical Applicability Module**: The ensemble learning model is integrated into a user-friendly interface that allows clinicians to input patient data and receive predictions for metastasis detection and survival time estimation.
- **Continuous Learning and Updating Module**: The system is designed for continuous learning, allowing it to adapt and improve over time as new data becomes available.

4.2 Detailed Design

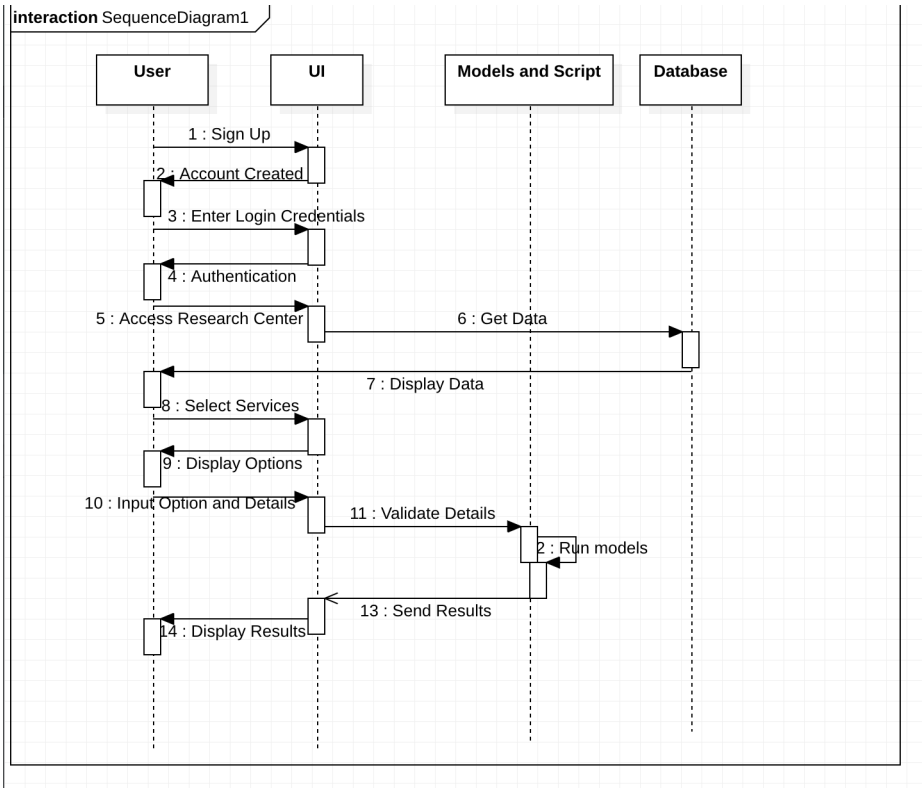

Fig 4.6

Sequence Diagram

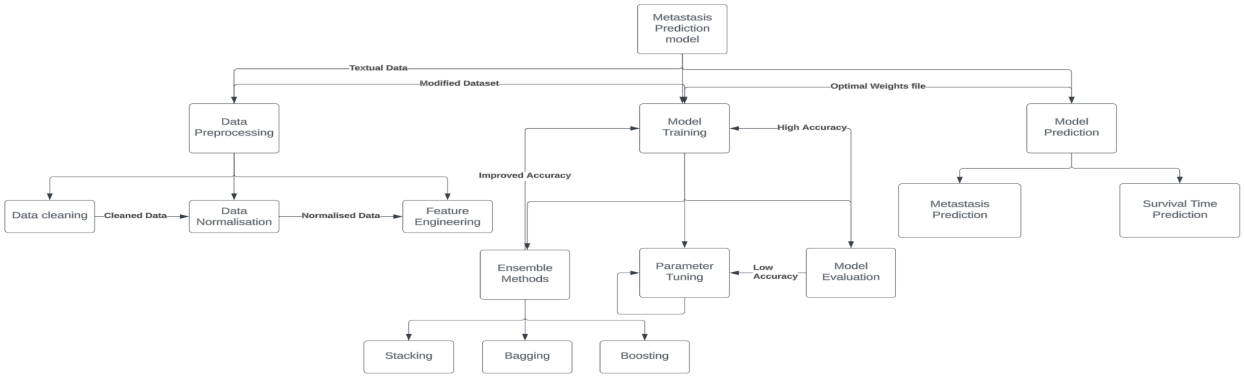

Fig 4.7 Structure Chart

4.2.1 Functional Description of Modules

Data Preprocessing:

This module is responsible for preparing the raw input data for further analysis and modeling. It involves the following sub-modules:

- Data Cleaning:

This sub-module focuses on identifying and handling missing, erroneous, or inconsistent data points. It involves techniques such as imputation, removal of outliers, and addressing data inconsistencies to ensure the quality and reliability of the data.

- Data Normalization:

In this sub-module, the data is transformed to a common scale or distribution. This helps in ensuring that different features contribute proportionally to the analysis, preventing certain features from dominating the results due to their larger magnitudes.

- Feature Engineering:

Feature engineering involves creating new features or modifying existing ones to improve the performance of machine learning models. This sub-module may include techniques such as feature selection, dimensionality reduction, and creating derived features from existing ones.

Model Training:

This module is responsible for training machine learning models using the preprocessed data. It involves the following sub-modules:

- Ensemble Methods:

Ensemble methods combine the predictions of multiple models to enhance predictive accuracy and robustness. This sub-module includes techniques like Random Forest, Gradient Boosting, and Bagging, which create an ensemble of weaker models to make accurate predictions.

- Parameter Tuning:

Parameter tuning involves optimizing the hyperparameters of machine learning algorithms to achieve the best model performance. This sub-module uses techniques like grid search or random search to find the optimal combination of hyperparameters.

- Model Evaluation:

This sub-module assesses the performance of the trained models using appropriate metrics such as accuracy, precision, recall, and F1-score. It involves techniques like cross-validation to ensure the models generalize well to new data.

Model Prediction:

This module is responsible for making predictions based on the trained models. It includes the following sub-modules:

- Metastasis Prediction:

This sub-module takes input data related to a diagnosed patient's history and predicts the likelihood of cancer metastasis. It uses the trained models to analyze the input features and provide an estimation of the risk of cancer spreading.

- Survival Time Prediction:

In this sub-module, input data about a patient diagnosed with Malignant Cancer is used to predict the approximate survival time. The trained models analyze the patient's characteristics and history to estimate the duration of survival.

CHAPTER 5

Implementation

5.1 Programming Language Selection

Python is a popular choice for machine learning projects due to its simplicity, extensive libraries, and active community support. It provides a wide range of tools and frameworks specifically designed for machine learning tasks, making it an ideal choice for your project. This is because:

Easy to Learn and Read: Python has a clean and intuitive syntax that is easy to understand, even for beginners. Its code readability and simplicity contribute to faster development and easier maintenance of machine learning projects.

Extensive Libraries: Python provides numerous libraries specifically designed for data manipulation, analysis, and machine learning. Some of the most popular libraries include NumPy, Pandas, Scikit-learn, TensorFlow, and PyTorch. These libraries offer pre-built functions and algorithms that simplify complex tasks and accelerate development.

Large Community and Documentation: Python has a vast and active community of developers and data scientists. This means you can find ample resources, tutorials, and code examples to help you overcome challenges and learn new techniques. Additionally, Python's extensive documentation makes it easier to understand and utilize its various libraries and frameworks effectively.

Data Manipulation and Analysis: Python's Pandas library provides powerful tools for data manipulation and analysis. It allows you to load data from various sources, clean and preprocess it, handle missing values, and perform exploratory data analysis. With Pandas, you can easily transform and reshape data, merge datasets, and extract meaningful insights.

Machine Learning Frameworks: Python offers popular machine learning frameworks like Scikit-learn, TensorFlow, and PyTorch. Scikit-learn provides a wide range of algorithms for tasks such as regression, classification, clustering, and dimensionality reduction. TensorFlow and PyTorch are deep learning frameworks that enable the creation and training of complex neural networks.

Visualization Capabilities: Python libraries like Matplotlib and Seaborn allow you to create rich visualizations to better understand your data and communicate your findings. These libraries provide a wide range of plots, charts, and graphs, enabling you to present your results effectively.

Integration and Extensibility: Python seamlessly integrates with other languages like C/C++ and Java, allowing you to leverage existing code and libraries. This integration provides flexibility in utilizing specialized libraries or optimizing computationally intensive parts of your project. Python also supports integration with databases, web frameworks, and APIs, making it suitable for various project requirements.

Deployment Options: Python provides multiple options for deploying machine learning models. You can integrate your models into web applications using frameworks like Flask or Django. Alternatively, you can leverage cloud platforms like AWS Lambda, Google Cloud Functions, or Microsoft Azure Functions to create serverless APIs for inference.

Some libraries used include:
NumPy: A fundamental library for scientific computing with Python, NumPy provides support for efficient numerical operations and multi-dimensional arrays, which are essential for handling data in machine learning projects.
Pandas: Pandas offers powerful data manipulation and analysis tools. It allows you to load, clean, preprocess, and transform your data, making it easier to work with various data formats and structures.
Scikit-learn: Scikit-learn is a versatile machine learning library that provides a wide range of algorithms and tools for tasks such as regression, classification, and model evaluation. It's beginner-friendly and widely used in the machine learning community.
Matplotlib: These libraries enable data visualization, allowing you to create informative plots, charts, and graphs to better understand your data and model performance.

HTML (Hypertext Markup Language) is the standard markup language for creating web pages. You can use HTML to structure and define the content of your web application. With Flask, you can render HTML templates to display dynamic content, such as model predictions or user input forms.

CSS (Cascading Style Sheets) is used to control the visual appearance of HTML elements. It allows you to define styles, colors, layouts, and other visual aspects of your web application. By integrating CSS with Flask, you can customize the look and feel of your application, making it more visually appealing and user-friendly.

Bootstrap is a popular CSS framework that provides a set of pre-designed components and styles for building responsive web applications. It offers ready-to-use templates, grid systems, navigation bars, buttons, and other UI elements. By incorporating Bootstrap with Flask, you can leverage its pre-built components to create a professional-looking interface for your model deployment.

5.2 Platform Selection

The Jupyter notebook was used as the IDE. Jupyter Notebook or JupyterLab is a popular choice for machine learning projects. It provides an interactive coding environment with support for inline documentation, data visualization, and easy code iteration. Jupyter notebooks allow you to combine code, visualizations, and explanations in a single document, making it easier to communicate your project findings.

Visual Studio Code (VS Code) is a powerful and versatile code editor that can greatly facilitate the development process for the machine learning project. VS Code provides excellent support for HTML, CSS, and JavaScript, with features like syntax highlighting, IntelliSense, code completion,

and error checking. You can create and edit your HTML, CSS, and JavaScript files directly in VS Code, benefiting from its robust code editing capabilities. The Live Server extension in VS Code allows you to run a local development server directly from the editor. It automatically refreshes your web page whenever you make changes to your HTML, CSS, or JavaScript files. This feature provides a real-time preview of your web application, enhancing your development workflow. VS Code's integrated terminal is valuable for web development as it allows you to run commands and scripts related to your web project. You can execute commands for package installation, build processes, or server setup directly within the terminal, saving you the hassle of switching to a separate terminal window. If your web development project involves version control with Git, VS Code offers seamless integration. You can manage your Git repositories, view changes, commit code, switch branches, and perform other Git operations directly from the editor. The Git integration ensures smooth collaboration and efficient management of your web project.

5.3 Coding Conventions

Code conventions refer to a set of guidelines and best practices for writing code that promote readability, maintainability, and consistency across a codebase. Code formatter can automatically format your code according to your configured code style settings. It helps maintain consistent indentation, spacing, and line wrapping to improve code readability.

PEP 8 Style Guide: PEP 8 (Python Enhancement Proposal 8) is the official style guide for Python code. It provides recommendations on how to format code, name variables, use indentation, and more. Following PEP 8 ensures that code is consistent and readable across different Python projects. Some key conventions include:

- Using four spaces for indentation.
- Limiting lines to a maximum of 79 characters.
- Using lowercase letters and underscores for variable and function names (snake_case).
- Using uppercase letters for constants.
- Using whitespace judiciously for improved readability.

Naming Conventions

Variables and Functions

- Use descriptive names that accurately convey the purpose or content of the variable or function.
- Use lowercase letters and underscores (snake_case) for variable and function names.

Classes:

- Use CamelCase for class names, starting with an uppercase letter.
- Class names should be nouns or noun phrases that describe the entity being represented.

Constants:

- Use uppercase letters for constant variable names.

Modules and Packages:

- Use lowercase letters and underscores for module and package names.

Arguments and Parameters:

- Use descriptive names for function arguments and parameters, following the same conventions as variables.
- The name of the arguments must convey their meaning.
- Docstrings should be used to indicate type and description of parameters.

Test Functions:

- Prefix test functions with "test_" to clearly indicate that they are test cases.
- Use descriptive names for the test functions that reflect the specific scenario being tested.

It's important to maintain consistency and clarity throughout your codebase. By following these naming conventions, you can improve the readability, understandability, and maintainability of your project.

File Organization

Organizing your project's file structure is essential for maintaining a clean and structured codebase. While there can be variations depending on project size, complexity, and specific requirements, here is a recommended file organization structure for your machine learning project involving predicting the chance of admission:

Root Directory:

- README.md: Project documentation and instructions.
- requirements.txt: File listing the project dependencies and their versions.
- .gitignore: File specifying which files and directories should be ignored by version control (e.g., virtual environment files, data files).

Source Code:
Create a dedicated directory for your source code, such as "src" or "app."
main.py: Entry point for your application or script that orchestrates the model prediction and interaction.

- data_preprocessing.py: Module for data preprocessing functions and pipelines.
- model.py: Module containing the machine learning model implementation and related functions.

- evaluation.py: Module for model evaluation metrics and functions.

Tests:
Create a separate directory for storing your test files, such as "tests" or "test_cases."
- test_data_preprocessing.py: Test cases for data preprocessing functions.
- test_model.py: Test cases for the machine learning model implementation.
- test_evaluation.py: Test cases for the evaluation metrics.

Data:
Include a directory to store the dataset or datasets used for training and testing the model.
- Optionally, you can create subdirectories to organize different datasets or versions of datasets.
- The cleaned datasets can also be put in the same directory.

Notebooks:
- If you plan to use Jupyter Notebooks for experimentation or analysis, create a directory to store your notebooks.
- Include separate notebooks for data exploration, model development, and evaluation.

Documentation:
- Create a directory to store project-specific documentation.
- Include any project-specific notes, documentation on data sources, or additional information.

Declarations

When it comes to declarations in Python, there are some conventions and best practices that are commonly followed. These conventions help make your code more readable, maintainable, and consistent. Here are some conventions for declarations in Python:

Variable Declarations:
- Use meaningful and descriptive names for variables that accurately represent their purpose.
- Use lowercase letters and underscores (snake_case) for variable names.
- Initialize variables with appropriate default values whenever possible.
- Declare variables close to their first usage to enhance code readability.

Function and Method Declarations:
- Use lowercase letters and underscores (snake_case) for function and method names.
- Choose descriptive names that indicate the purpose or action performed by the function.
- Use verb-noun combinations or verb phrases for function and method names.
- Separate multiple words in function and method names with underscores.

Constant Declarations:

- Use uppercase letters for constant names.
- Separate multiple words in constant names with underscores.
- Constants should represent fixed values that do not change during program execution.

Class Declarations:

- Use CamelCase for class names, starting with an uppercase letter.
- Class names should be nouns or noun phrases that describe the entity being represented.
- Make classes as self-contained and cohesive as possible, following the Single Responsibility Principle.

Module and Package Declarations:

- Use lowercase letters and underscores for module and package names.
- Module names should be short, descriptive, and related to the functionality they provide.
- Package names should be meaningful and reflect the purpose of the collection of modules.

Import Declarations:

- Import modules on separate lines, rather than using comma-separated imports on a single line.
- Use absolute imports when importing from other modules in your project.
- Group imports into sections: standard library imports, third-party library imports, and local imports.
- Avoid using wildcard imports (from module import *) as they can pollute the namespace and make code less maintainable.

Constant Declarations:

- Use uppercase letters for constant names.
- Separate multiple words in constant names with underscores.
- Constants should represent fixed values that do not change during program execution.

Comments

Writing effective comments is crucial for improving code comprehension and facilitating collaboration among developers. Here are some conventions and best practices for writing comments in Python:

- Use Complete Sentences: Write comments in complete sentences, using proper grammar and punctuation. This helps ensure clarity and readability.
- Comment Purpose and Intent: Clearly explain the purpose, intent, or rationale behind the code. Describe why something is being done, especially if the code logic is not self-evident. Comments should provide context and help other developers understand the code's intention.

- Avoid Redundant Comments: Avoid commenting on obvious or self-explanatory code. Comments should add value by providing additional insights, explaining complex logic, or clarifying assumptions or design choices.
- Use Inline Comments Sparingly: Use inline comments sparingly and only when necessary. Inline comments are comments placed on the same line as the code and are generally used to explain specific lines or sections of code that might be unclear.
- Comment Formatting: Format comments to improve readability and distinguish them from the code. Use consistent indentation and spacing to align comments with the surrounding code. Consider using whitespace to separate comments from the code for clarity.
- Update and Maintain Comments: Keep comments up to date as code evolves. Whenever you make changes to the code, review and update related comments to reflect the current state of the code. Outdated comments can be misleading and lead to confusion.

Document Functionality and Parameters: Document functions and methods using docstrings, which are multi-line comments enclosed in triple quotes (""""). Describe the functionality of the function, its parameters, return values, and any exceptions it might raise. Follow the appropriate docstring style conventions, such as the Google Docstring Style or reStructuredText.

Docstrings are multi-line comments enclosed in triple quotes (""") that provide documentation for files, modules, classes, functions, and methods in Python. They serve as a valuable tool for describing the purpose, functionality, usage, and other important details of code entities. Here's a note on docstrings used for describing files and functions:

File-level docstrings provide an overview of the entire file or module. They are typically placed at the beginning of the file, immediately after the import statements, and before any class or function definitions. File-level docstrings often include the following information:

- File Summary: Describe the purpose and high-level functionality of the file or module.
- Author Information: Provide the name of the author or the team responsible for the file.
- Date and Version: Optionally, include the creation date and version information.
- Dependencies: List any external dependencies or libraries required by the file.

Usage Examples: Include examples or usage scenarios for the file if applicable.

Function docstrings provide detailed documentation for individual functions or methods. They are placed immediately after the function or method definition and before any statements within the function body. Function docstrings often include the following information:

Function Summary: Provide a concise summary of the function's purpose and functionality.
- Parameters: List the parameters accepted by the function, along with their types, descriptions, and any default values.
- Return Value: Describe the data type and meaning of the value returned by the functions
- Exceptions: If the function raises specific exceptions, document them and explain the circumstances under which they might occur.

- Usage Examples: Include examples of how to use the function with different parameter values and expected outputs.

Side Effects: If the function has any notable side effects, such as modifying global variables or writing to files, document them.

5.4 Summary

The implementation module is a crucial phase in software development, encompassing various sub-modules that collectively transform design concepts into functional software. Coding conventions establish consistent coding practices, enhancing code readability and maintainability. Platform selection involves choosing the environment or framework that aligns with project requirements, ensuring scalability and compatibility. Programming language selection determines the coding efficiency and capabilities of the application. These sub-modules collaboratively contribute to the successful translation of design and conceptualization into a functional software solution, fostering a structured and efficient implementation process.

CHAPTER 6

Experimental Results and Testing

6.1. Evaluation Metrics

If classification models are used, the following metrics are relevant:

● Accuracy: Accuracy measures the proportion of correct predictions over the total number of predictions. It is the most common and straightforward metric for classification tasks but may not be suitable if the classes are imbalanced.

● Precision: Precision quantifies the proportion of correctly predicted positive instances (metastasis detected) out of all instances predicted as positive. It focuses on the accuracy of positive predictions and is useful when false positives are costly.

● Recall (Sensitivity or True Positive Rate): Recall measures the proportion of correctly predicted positive instances (admission) out of all actual positive instances. It emphasizes the ability of the model to find all positive instances and is important when false negatives are costly.

●F1 Score: The F1 score combines precision and recall into a single metric. It provides a balanced measure of both metrics and is useful when precision and recall are both important.

●Area Under the ROC Curve (AUC-ROC): ROC (Receiver Operating Characteristic) curve is a plot of true positive rate (TPR) against the false positive rate (FPR) at various classification thresholds. AUC-ROC represents the overall performance of the model across different thresholds. It is commonly used when there is a need to evaluate the model's performance at various operating points.

If you treat the prediction problem as a regression task (e.g., predicting a continuous chance of admission), the following metrics are commonly used:

●Mean Squared Error (MSE): MSE calculates the average squared difference between the predicted and actual values. It measures the overall quality of the model's predictions, with higher values indicating greater errors.

●Root Mean Squared Error (RMSE): RMSE is the square root of the MSE. It provides a more interpretable metric by presenting the error in the original unit of the target variable.

● Mean Absolute Error (MAE): MAE calculates the average absolute difference between the predicted and actual values. It provides a measure of the average magnitude of errors without considering their direction.

●R-squared (Coefficient of Determination): R-squared represents the proportion of the variance in the target variable that can be explained by the model. It ranges from 0 to 1, where 1 indicates a perfect fit.

● Mean Absolute Percentage Error (MAPE): MAPE measures the average percentage difference between the predicted and actual values. It provides insights into the average magnitude of errors relative to the actual values.

6.2. Experimental Dataset

The experimental dataset used for this project is sourced from the Surveillance, Epidemiology, and End Results (SEER) database. SEER is a comprehensive cancer database that collects data on cancer incidence, prevalence, and survival from various geographic regions across the United States. The dataset contains a wealth of information about cancer patients, including demographics, tumor characteristics, treatment details, and survival outcomes. Dataset contains roughly 100,000 records with 14 attributes. This dataset is further split in 70:20:10 proportion into training,testing and validation datasets respectively.

6.3. Performance Analysis

6.3.1 Metastasis Prediction

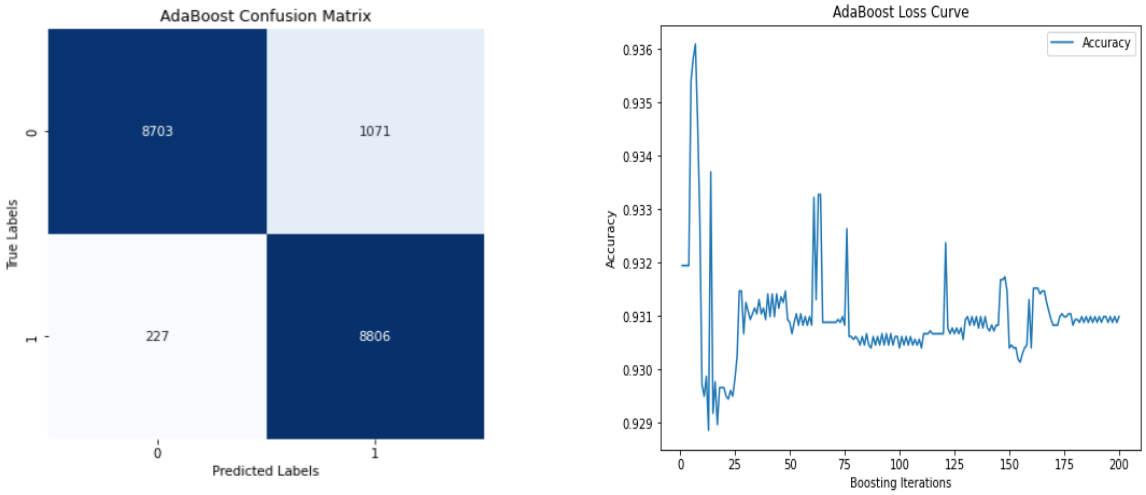

Fig 6.1 Confusion matrix and loss curves for AdaBoost

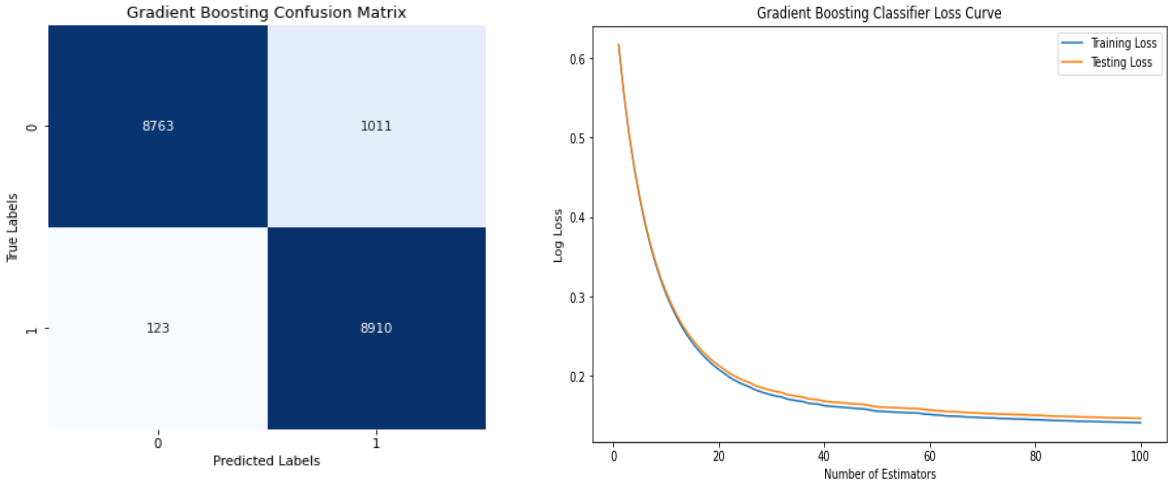

Fig 6.2 Confusion matrix and loss curves for GBC

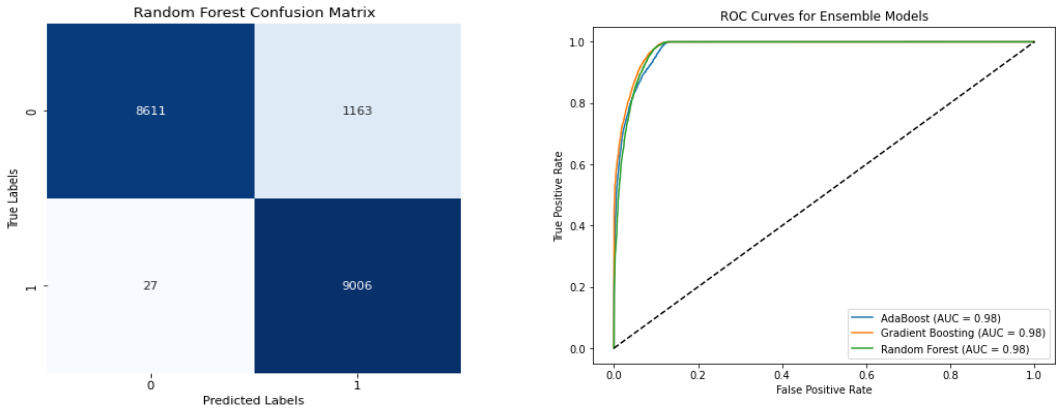

Fig 6.3 Confusion matrix for random forest and AUC-ROC curves

Model	Accuracy	Precision	Recall	F1-Score
Random Forest	0.937	0.893	0.996	0.938
GBC	0.929	0.89	0.983	0.94
AdaBoost	0.931	0.891	0.974	0.931

Table 6.1 Evaluation Metrics Comparison

```
print(rf.feature_importances_*100)
```

```
[ 0.23767731  6.46894435  3.7560089   0.42062049  5.75984437  9.46945154
  3.0793008   0.37711303 65.44553748  2.39476278  2.59073895]
```

Fig 6.4 Important Attributes for Random Forest

It was observed that most important attribute was **RXSummSurgPrimSite(65%)**

7.3.2 Survival Time

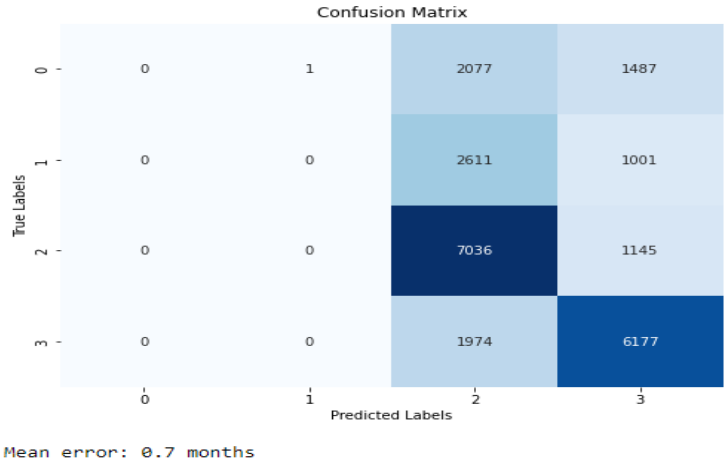

Fig 6.5 Confusion matrix for AdaBoost

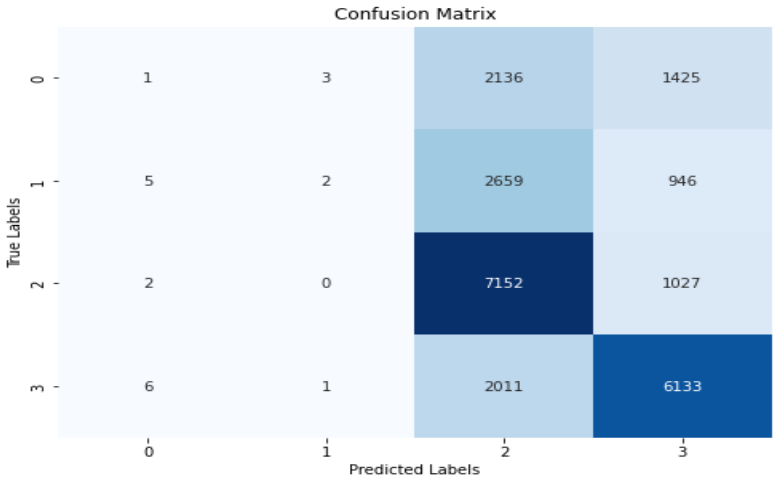

Fig 6.6 Confusion Matrix for GBC

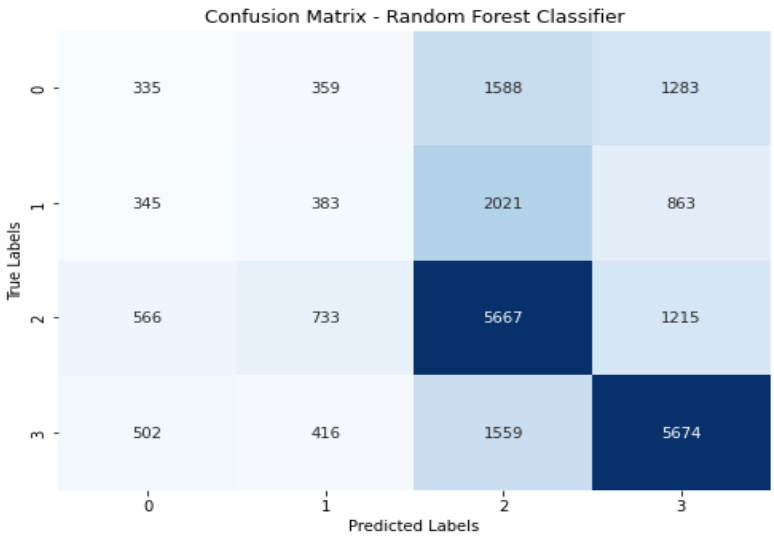

Fig 6.7 Confusion Matrix for Random Forest

Model	Training Accuracy	Testing Accuracy	Mean Error
GBC	0.72	0.61	0.69
AdaBoost	0.57	0.56	0.7
Random Forest	0.71	0.51	0.79

Table 6.2 Evaluation Metrics Comparison

```
print(gbc.feature_importances_*100)
```

```
[ 4.97214844  9.30184872 10.14129665  3.11310448  6.49002511  3.46164481
 19.87990428 37.47722871  1.55246036  3.61033843]
```

Fig 6.8 Important attributes for GBC

It was observed that the most important attributes were **RXSummSurgPrimSite(37%)** and **RXSummScopeRegLNSur2003(19%)**

6.4 Unit Testing

Unit testing can be done using Pytest. When it comes to unit testing in a machine learning project involving predicting the chance of admission, you can write several unit tests to ensure the correctness and robustness of your code.

Key features of Pytest include:

●Test Discovery: Pytest automatically discovers and runs tests based on naming conventions and directory structures. It identifies test files that start with "test_" or end with "test.py" and recognizes test functions prefixed with "test".
●Fixture Support: Fixtures are functions that provide a baseline set of data or objects to be used in tests. Pytest allows the creation of fixtures, which can set up and tear down test environments, simulate test data, or establish connections to external resources. Fixtures promote code reusability and help maintain clean and modular test code.
●Assertions: Pytest provides a rich set of built-in assert statements for making assertions about expected results. It includes standard assertions for equality, inequality, containment, and more. Additionally, Pytest generates informative failure messages that highlight the exact point of failure, making it easier to diagnose and fix issues.
●Parameterization: Pytest allows tests to be parameterized, enabling the execution of a single test function with multiple sets of inputs. This simplifies the testing process by reducing code duplication and providing a concise way to test multiple scenarios.

●Test Coverage: Pytest integrates well with coverage measurement tools, allowing you to measure the code coverage of your tests. By using plugins like pytest-cov, you can generate coverage reports that indicate which parts of your codebase are exercised by the tests, helping you identify areas that need more thorough testing.

The following checks had to be covered:

- Check all the GUI elements for size, position, width, length, and acceptance of characters or numbers. For instance, you must be able to provide inputs to the input fields.
- Check if an individual can execute the intended functionality of the application using the GUI
- Check for Clear demarcation of different sections on screen
- Check Font used in an application is readable
- Check the alignment of the text is proper
- Check the Color of the font and warning messages is aesthetically pleasing
- Check that the images have good clarity

6.5 Integration Testing

Integration testing focuses on verifying that different parts of the system work correctly together and that data flows and transformations are handled properly. Here are some integration testing scenarios to consider:

- Data Preprocessing Integration Testing: Test the integration between data preprocessing steps, such as handling missing values, scaling features, or encoding categorical variables. Ensure that the data preprocessing pipeline functions as expected and produces the desired transformations on the input data.
- Model Training and Evaluation Integration Testing: Test the integration between the model training and evaluation components. Verify that the model is trained on the correct input data and that the evaluation metrics are calculated accurately. This includes checking the consistency of data splits for training and testing, as well as verifying the correctness of the model training process.
- Input Data Integration Testing: Test the integration between the input data handling and the prediction pipeline. Ensure that the input data is correctly processed, and any necessary transformations, such as feature scaling or encoding, are applied consistently during both training and prediction stages.
- API Integration Testing: If you have an API or web service for interacting with your machine learning model, perform integration testing to ensure that the API endpoints, request handling, and response generation are working correctly. Test various scenarios, including valid and invalid inputs, edge cases, and error handling.
- Deployment and Infrastructure Integration Testing: If your model is deployed on a specific infrastructure or platform, perform integration testing to ensure the seamless integration and functionality of the deployed system. Test the interaction between the deployed model and any supporting components, such as web servers, databases, or third-party services.
- End-to-End Integration Testing: Conduct end-to-end integration testing to verify the entire workflow of your machine learning project. This includes testing the complete pipeline from data ingestion to prediction generation. Validate that the input data is processed correctly, the model is trained, and predictions are produced accurately.

During integration testing, it is crucial to create test cases that cover various scenarios, including both normal and exceptional conditions. This helps identify any issues or inconsistencies in the interactions between different components of the system. Additionally, it is essential to ensure the integrity of the test environment and provide consistent and representative test data.

6.6 System Testing

System testing for lung cancer metastasis and survival time prediction involves evaluating the developed models and system components to ensure their accuracy, reliability, and effectiveness in real-world scenarios.

- Gather a diverse and representative test dataset that mirrors real clinical scenarios. Preprocess the test data similarly to the training data, ensuring consistency.
- Metastasis Prediction Testing: Input test data into the metastasis prediction model (e.g., ensemble learning model). Evaluate the model's predictions against ground truth metastasis labels. Calculate and analyze metrics such as accuracy, precision, recall, and F1-score. Visualize the model's performance using confusion matrices and ROC curves.
- Survival Time Prediction Testing: Input test data into the survival time prediction model (e.g., survival analysis model). Compare predicted survival times with actual survival times. Use metrics like concordance index (C-index) to assess survival prediction accuracy. Plot Kaplan-Meier survival curves for visual comparison.
- Integrated Testing: Combine the metastasis and survival time prediction models into the integrated system. Use real patient data to simulate the entire process of predicting metastasis and survival time. Evaluate how well the system works as a whole, considering interdependencies between components. Clinical Applicability Testing: Collaborate with oncologists and medical experts to validate the system's predictions and recommendations. Gather feedback on the system's usability, interpretability, and usefulness in clinical decision-making.
- Performance and Scalability Testing: Test the system's performance with varying dataset sizes and complexities. Evaluate processing times and resource utilization to ensure scalability.
- Robustness Testing: Introduce noise, outliers, and missing values to the test dataset to assess the model's robustness. Analyze how well the system handles unexpected data variations.
- Documentation and Reporting: Document testing procedures, methodologies, and results. Prepare a comprehensive report summarizing the testing outcomes and insights gained. Provide recommendations for improvements and further research.
- Iterative Improvement: Analyze the testing outcomes and user feedback to identify areas for improvement. Refine models, algorithms, and system components based on testing insights. Repeat testing iterations as needed to achieve desired performance and reliability. System testing ensures that your lung cancer metastasis and survival time prediction system is accurate, reliable, and beneficial to both clinicians and patients.

CHAPTER 7

Conclusion and Future Enhancement

7.1 Limitations of Project

1. Complexity of Cancer Progression - Due to heterogeneity, Lung cancer is a complex disease with various subtypes and factors affecting its progression. Our Models do not account for **all** potential factors contributing to metastasis and survival time accurately.
2. External Factors - Patient outcomes can be influenced by factors beyond the scope of medical data, such as socioeconomic conditions, lifestyle, and access to healthcare.
3. Clinical Variability - Clinical diagnoses and practices can vary across medical professionals and institutions, affecting the accuracy of the input data and predictions and cannot be captured effectively in datasets.
4. Subset of Biomedical indicators - The dataset consists of only certain indicators and is still missing some important factors, learned from literature.
5. UI design - The design is intuitive and easy to use, but isn't scalable enough. It doesn't have the required front-end tools to host the website to actually put it into use, yet.

7.2 Conclusion and Future Enhancements

In conclusion, the development of the Lung Cancer Detection System marks a significant step toward improving diagnostic accuracy and patient outcomes in the field of oncology. By leveraging advanced machine learning techniques on textual data from the SEER Research database, the system aims to predict metastasis likelihood, estimate survival time, and identify key attributes contributing to cancer progression. Some of the general ideas we would like to incorporate into the pipeline and functionality in the future iterations are as follows:

1. Incorporating Clinical Data: Integrating clinical data such as patient medical history, treatment regimens, and biomarker profiles can provide a more comprehensive view of each patient's condition, potentially improving prediction accuracy.

2. Deep Learning for Text: Exploring deep learning models, such as recurrent neural networks (RNNs) and transformers, can help capture the intricate relationships within medical text data, enabling more accurate predictions.
3. External Data Sources: Integrating external data sources, such as genomics data or environmental factors, can enrich the dataset and enhance the system's predictive power.

7.3 Summary

The Lung Cancer Detection System is a vital development with the potential to revolutionize oncology diagnosis and patient care. By leveraging advanced machine learning techniques on textual data from the SEER Research database, the system aims to predict metastasis likelihood, estimate survival time, and identify crucial factors in cancer progression. Looking ahead, the system can further evolve and improve through various avenues. These include incorporating clinical data for a holistic view, exploring deep learning models for text analysis, integrating multimodal data, enabling real-time monitoring, and enhancing interpretability.