High Level Design Document

Project Name: Mushroom Classification

Website Live Link: *Click Here*
Revision No: 1

Name: Chetan Mahale
Email: chetan.mahale0220@gmail.com

Table of Contents

Abstract

The Audubon Society Field Guide to North American Mushrooms provides comprehensive descriptions of 23 species of gilled mushrooms within the Agaricus and Lepiota family. Each species is classified into two categories: definitely edible or toxic (which includes mushrooms that are either definitely poisonous or maybe edible but not recommended). This consolidation reflects the guide's emphasis on caution, underscoring the complexity and potential risks involved in mushroom identification. Unlike some plants that can be easily identified as safe or hazardous by simple rules, such as the "leaflets three, let it be" adage for Poisonous Oak and Ivy, mushrooms lack straightforward indicators for determining their edibility.

The primary objective is to develop a reliable method for predicting whether a given mushroom is poisonous or edible. Given the diverse and often subtle characteristics that differentiate edible mushrooms from toxic ones, the challenge lies in creating an accurate and practical predictive model. Such a model would significantly enhance safety for foragers and enthusiasts by reducing the risk of accidental poisoning. This study aims to leverage the detailed descriptions and classifications provided in the guide to construct a predictive framework, thereby addressing the guide's assertion that no simple rule exists for judging mushroom edibility.

Introduction

A high-level design document is crucial for this project because it provides clarity and direction, ensuring that all team members and stakeholders have a shared understanding of the project's objectives, requirements, and constraints. This common understanding helps align the efforts towards a common goal and reduces the risk of misunderstandings. Additionally, the document offers a structured approach, breaking down complex processes into manageable components and defining the architecture, key modules, and interactions between different parts of the system. This organisation facilitates better planning, resource allocation, and scheduling, ensuring that each phase of the project is well-coordinated and efficient.

Moreover, the high-level design document aids in risk management by identifying potential risks and obstacles early in the project. This allows for proactive risk mitigation strategies, ensuring smooth project progression without major disruptions. The document also facilitates collaboration and review, serving as a reference point for team members and stakeholders to provide feedback and ensure the design's robustness. Furthermore, it lays the groundwork for detailed design and implementation phases, guiding the creation of technical specifications, coding, testing, and deployment plans. Finally, the document serves as essential documentation for future reference, aiding in maintenance, scalability, and the onboarding of new team members, ensuring the project's long-term sustainability.

Chapter 1
Description

1.1 Problem Perspective

The problem at hand involves accurately predicting the edibility of mushrooms, specifically distinguishing between those that are definitely edible and those that are toxic, within the Agaricus and Lepiota families as described in The Audubon Society Field Guide to North American Mushrooms. This is a critical issue because the consequences of misidentification can be severe, potentially leading to poisoning and serious health risks. The complexity of this task is compounded by the fact that there are no simple, universally applicable rules to determine mushroom edibility, as subtle morphological differences can signify whether a mushroom is safe to consume or dangerously toxic. Thus, the primary challenge lies in developing a reliable predictive model that can accurately classify mushrooms based on their detailed characteristics, enhancing the safety and confidence of foragers and enthusiasts in identifying edible species. This project aims to leverage detailed species descriptions and classification data to create such a model, addressing the inherent risks and uncertainties in mushroom identification.

1.2 Problem Statement

The Audubon Society Field Guide to North American Mushrooms contains descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota Family Mushroom (1981). Each species is labelled as either definitely edible, definitely poisonous, or maybe edible but not recommended. This last category was merged with the toxic category. The Guide asserts unequivocally that there is no simple rule for judging a mushroom's edibility, such as "leaflets three, leave it be" for Poisonous Oak and Ivy. The main goal is to predict which mushroom is poisonous & which is edible.

1.3 Project Solution

The solution to this project involves developing a machine learning-based predictive model that can accurately classify mushrooms in the Agaricus and Lepiota families as either edible or toxic. The process begins with data collection, utilising the detailed descriptions and classifications provided in The Audubon Society Field Guide to North American Mushrooms. This data will be preprocessed to extract relevant features such as morphological characteristics, habitat, and other distinguishing factors.
Once the data is prepared, it will be split into training and testing sets to build and evaluate the model. Various machine learning algorithms, such as Logistic Regression, SVM, KNN, Naive Bayes, Decision Trees, and Artificial Neural Networks will be explored to determine the most effective approach for classification. The model's performance will be evaluated using metrics like accuracy, precision, recall, and F1-score to ensure robust and reliable predictions.

Feature selection and engineering techniques will be employed to enhance the model's predictive power by identifying the most significant attributes that differentiate edible mushrooms from toxic ones.
Additionally, the solution will include a user-friendly interface or application where users can input characteristics of a mushroom to receive an immediate prediction of its edibility status. This interface will also provide explanations for the predictions, offering insights into which features influenced the classification decision.

1.4 Technical Requirement

The technical requirements for this project can be divided into several categories, encompassing data requirements, model development, software and tools, user interface, and testing and validation. Each category ensures that the project is executed efficiently, resulting in a robust and reliable predictive model.

Data Requirements:

1. Data Source: *[Link]*
   Detailed descriptions of 23 species of gilled mushrooms in the Agaricus and Lepiota families from The Audubon Society Field Guide to North American Mushrooms.
2. Data Features: Key morphological characteristics include:

   - cap-shape:
     bell=b, conical=c, convex=x, flat=f, knobbed=k, sunken=s

   - cap-surface:
   - fibrous=f, grooves=g, scaly=y, smooth=s

   - cap-colour:
     brown=n, buff=b, cinnamon=c, grey=g, green=r, pink=p, purple=u, red=e, white=w, yellow=y

   - bruises:
     bruises=t, no=f

   - odor:
     almond=a, anise=l, creosote=c, fishy=y, foul=f, musty=m, none=n, pungent=p, spicy=s

   - gill-attachment:
     attached=a, descending=d, free=f, notched=n

   - gill-spacing:
     close=c, crowded=w, distant=d

   - gill-size:
     broad=b, narrow=n

- gill-colour:
  black=k, brown=n, buff=b, chocolate=h, gray=g, green=r, orange=o, pink=p, purple=u, red=e, white=w, yellow=y

- stalk-shape:
  enlarging=e, tapering=t

- stalk-root:
  bulbous=b, club=c, cup=u, equal=e, rhizomorphs=z, rooted=r, missing=?

- stalk-surface-above-ring:
  fibrous=f, scaly=y, silky=k, smooth=s

- stalk-surface-below-ring:
  fibrous=f, scaly=y, silky=k, smooth=s

- stalk-colour-above-ring:
  brown=n, buff=b, cinnamon=c, gray=g, orange=o, pink=p, red=e, white=w, yellow=y

- stalk-colour-below-ring:
  brown=n, buff=b, cinnamon=c, gray=g, orange=o, pink=p, red=e, white=w, yellow=y

- veil-type:
  partial=p, universal=u

- veil-colour:
  brown=n, orange=o, white=w, yellow=y

- ring-number:
  none=n, one=o, two=t

- ring-type:
  cobwebby=c, evanescent=e, flaring=f, large=l, none=n, pendant=p, sheathing=s, zone=z

- spore-print-colour:
  black=k, brown=n, buff=b, chocolate=h, green=r, orange=o, purple=u, white=w, yellow=y

- population:
  abundant=a, clustered=c, numerous=n, scattered=s, several=v, solitary=y

- habitat:
  grasses=g, leaves=l, meadows=m, paths=p, urban=u, waste=w, woods=d

3. Data Preprocessing: Techniques to clean, normalise, and encode categorical data for machine learning algorithms.

Model Development:

1. Algorithms: Implementation of various machine learning algorithms such as Logistic Regression, SVM, KNN, Naive Bayes, Decision Trees, and Artificial Neural Networks to determine the best-performing model.
2. Feature Engineering: Identification and creation of significant features that improve model accuracy.
3. Model Training: Splitting data into training and testing sets, ensuring balanced representation of edible and toxic mushrooms.
4. Evaluation Metrics: Use of accuracy, precision, recall, F1-score, and confusion matrix to evaluate model performance.

Software and Tools:

1. Programming Languages: Python for data analysis and model development.
2. Libraries and Frameworks: Scikit-learn, TensorFlow, Keras, Pandas, NumPy, and Matplotlib for machine learning, data manipulation, and visualisation.
3. Development Environment: Jupyter Notebook or Integrated Development Environment (IDE) like VSCode for coding and experimentation.

User Interface:

1. Application Development: Creation of a user-friendly interface, possibly a website, where users can input mushroom characteristics and receive edibility predictions.
2. Frameworks: Use of streamlit framework for website creation.
3. Explanation Module: Integration of a module that provides explanations for the model's predictions, enhancing user understanding and trust.

Deployment and Maintenance:

1. Cloud Services: Utilisation of streamlit's cloud platform for model deployment and scaling.

By adhering to these technical requirements, the project will be well-equipped to develop a reliable and practical predictive model for mushroom edibility, ultimately enhancing safety and confidence for mushroom foragers and enthusiasts.

1.5 Constraints

Several constraints need to be considered in the development of the mushroom edibility prediction project:

1. Data Availability: The project relies heavily on the availability and quality of data from The Audubon Society Field Guide to North American Mushrooms. Constraints may arise if the data is limited, incomplete, or inaccurate, potentially affecting the performance and reliability of the predictive model.

2. Feature Selection: The project may face constraints in selecting relevant features for the predictive model. Certain features described in the field guide may not be easily measurable or may have limited variability, impacting the model's ability to distinguish between edible and toxic mushrooms accurately.

3. Model Performance: The accuracy and reliability of the predictive model are crucial constraints. The model must achieve high levels of precision, recall, and overall accuracy to ensure the safety of users relying on its predictions. Constraints may arise if the model fails to meet these performance metrics.

4. Computational Resources: The computational complexity of training and testing machine learning models, especially with large datasets and complex algorithms, can be a significant constraint. Limited computational resources may impact the speed and efficiency of model development and evaluation.

5. Regulatory Compliance: The project must adhere to regulatory guidelines and ethical considerations related to food safety and data privacy. Compliance with regulations such as the Food and Drug Administration (FDA) guidelines for food safety may impose constraints on the project's methodologies and deployment strategies.

6. User Interface Design: Constraints may arise in designing a user-friendly interface that effectively communicates the model's predictions and explanations to users. Ensuring accessibility, clarity, and ease of use for individuals with diverse backgrounds and expertise levels is essential.

7. Validation and Testing: Adequate validation and testing of the predictive model are necessary constraints to ensure its reliability and generalizability. The project must validate the model's predictions in real-world scenarios and incorporate feedback from domain experts and users to refine and improve its performance.

8. Deployment Environment: Constraints related to the deployment environment, such as compatibility with different operating systems, scalability, and maintenance requirements, must be considered to ensure seamless integration and long-term sustainability of the project.

By addressing these constraints proactively and iteratively throughout the project lifecycle, the mushroom edibility prediction project can overcome challenges and deliver a robust, accurate, and user-friendly solution for identifying edible and toxic mushrooms.

1.6 Assumptions

Several assumptions can be made for the mushroom edibility prediction project:

1. Data Accuracy: It is assumed that the descriptions and classifications of mushrooms provided in The Audubon Society Field Guide to North American Mushrooms are accurate and reliable. Any inaccuracies or discrepancies in the data could affect the performance of the predictive model.

2. Representativeness of Data: The data from the field guide is assumed to be representative of the broader population of mushrooms in the Agaricus and Lepiota families found in North America. While the guide may cover a diverse range of species, there may still be variations or rare species not accounted for in the data.

3. Generalizability of Model: The predictive model developed using the data from the field guide is assumed to generalise well to unseen mushrooms outside the dataset. This assumption is based on the belief that the features used for classification are indicative of edibility or toxicity across different species.

4. Consistency of Edibility Characteristics: It is assumed that certain morphological characteristics described in the field guide, such as cap shape, colour, and odour, are consistent indicators of mushroom edibility or toxicity. While these characteristics may hold true for many species, there may be exceptions or variations that the model needs to account for.

5. User Understanding: The users of the predictive model are assumed to have a basic understanding of mushroom identification and the associated risks of consuming wild mushrooms. The model's predictions and explanations are designed to supplement users' knowledge and assist in decision-making rather than replace expert judgement entirely.

Chapter 2: Design

2.1 Process Flow

Proposed Methodology

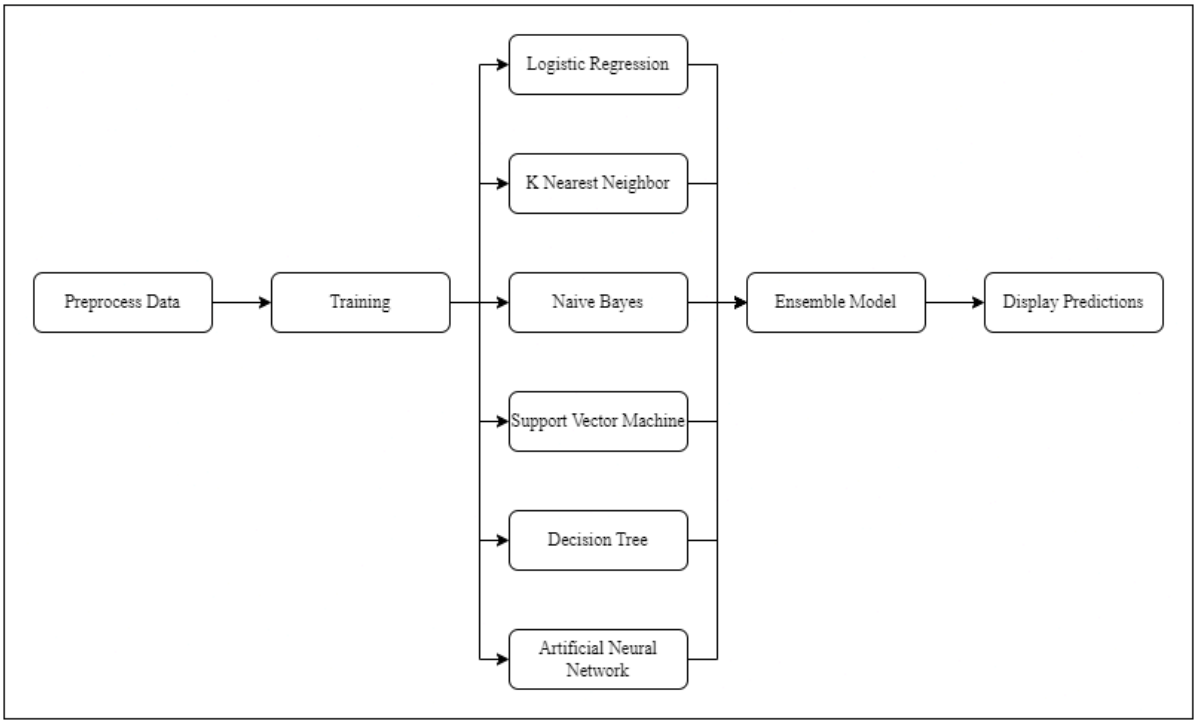

Fig: Proposed Methodology

Here is a detailed description of the process flow:

1. Preprocess Data

- The initial step involves preprocessing the raw input data. This includes tasks such as:
- Handling missing values
- Encoding categorical variables
- Normalising or scaling features
- Splitting the data into training and testing sets

2. Training

- The preprocessed data is then used to train multiple machine learning models. Each model is trained on the same dataset to ensure consistency in comparison. The models trained are:
- Logistic Regression
- K Nearest Neighbor (KNN)
- Naive Bayes
- Support Vector Machine (SVM)
- Decision Tree
- Artificial Neural Network (ANN)

3. Ensemble Model

- The trained models are combined into an ensemble model. The ensemble method typically improves the overall performance by leveraging the strengths of each individual model. The ensemble model in this scenario uses the Artificial Neural Network (ANN) as the final estimator to make the ultimate predictions.

4. Display Predictions

- The final predictions made by the ensemble model are displayed to the user. This step involves:
    - Receiving user input via the web interface
    - Preprocessing the user input similarly to the training data
    - Feeding the processed input to the ensemble model
    - Displaying the prediction results back to the user in an understandable format

High-Level Steps

1. User Interface (Frontend)

- User provides input data through a web form.

2. Backend

- The backend processes the input data, ensuring it is in the correct format for the models.
- The processed data is passed through the ensemble model.

3. Prediction and Display

- The ensemble model makes a prediction based on the input data.
- The prediction result is sent back to the frontend and displayed to the user.

Model training and evaluation

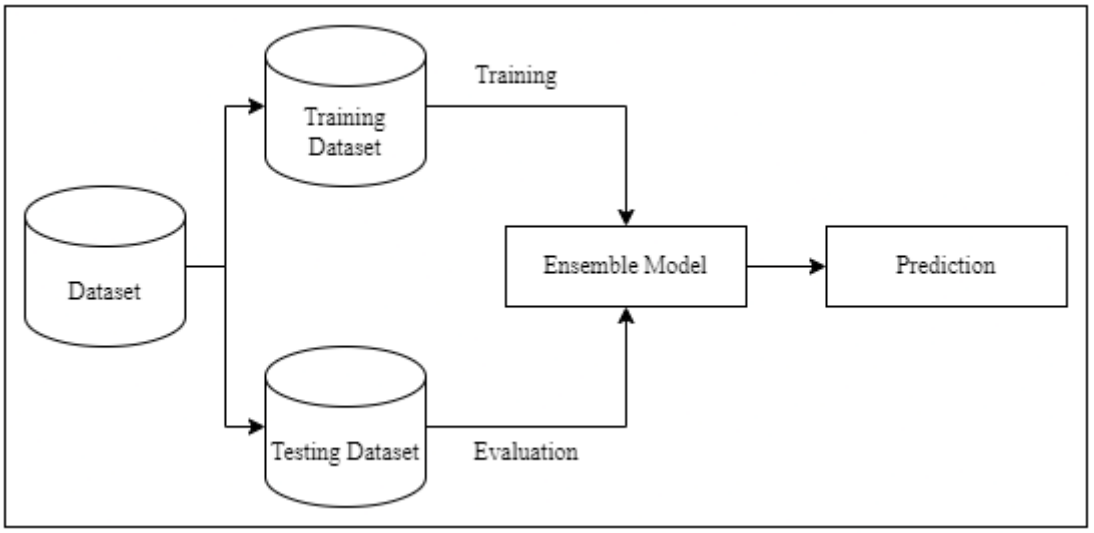

Fig: Training and Evaluation

Training

1. Dataset Collection:
- The process begins with collecting a comprehensive dataset that includes all the relevant features needed for training and evaluation.

2. Dataset Splitting:
- The dataset is split into two main parts: the Training Dataset and the Testing Dataset.
- The Training Dataset is used to train the ensemble model.
- The Testing Dataset is reserved for evaluating the performance of the trained model.

3. Training Dataset:
- The Training Dataset is fed into various machine learning models as part of the ensemble.
- These models may include Logistic Regression, K-Nearest Neighbour, Naive Bayes, Support Vector Machine, Decision Tree, and an Artificial Neural Network.
- Each model is trained on the Training Dataset to learn patterns and make predictions.

4. Ensemble Model:
- The trained models are combined into an Ensemble Model.
- The Ensemble Model leverages the strengths of individual models to improve overall performance and accuracy.
- The final estimator in the ensemble model is the Artificial Neural Network (ANN), which aggregates the predictions from the other models.

Evaluation

1. Testing Dataset:
- The Testing Dataset is used to evaluate the performance of the trained Ensemble Model.
- This dataset is separate from the training data to ensure unbiased evaluation.

2. Evaluation:
- The Ensemble Model makes predictions on the Testing Dataset.
- Various performance metrics, such as accuracy, precision, recall, and F1 score, are calculated to assess the model's performance.

3. Prediction:
- Based on the evaluation, the Ensemble Model is fine-tuned if necessary.
- The final predictions are made using the optimised Ensemble Model, which is then ready to be deployed in a web application.

Diagram Explanation

1. Dataset:
- Represents the initial dataset containing all the features and labels.

2. Training Dataset:
- A subset of the dataset used to train the ensemble model.

3. Testing Dataset:
- Another subset of the dataset used to evaluate the performance of the trained ensemble model.

4. Ensemble Model:
- Combines predictions from multiple models to improve overall accuracy and robustness.

5. Prediction:
- The final step where the ensemble model makes predictions based on the input data.

This high-level diagram and the accompanying explanation provide a clear understanding of the training and evaluation process for the ensemble model used in the web application.

Deployment process

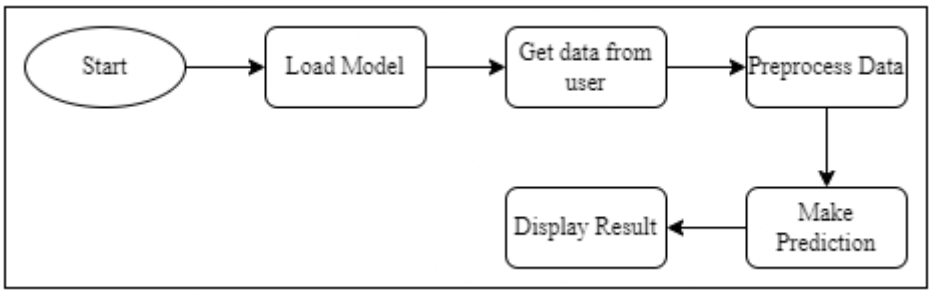

Fig: Deployment Process

1. Start
- The process begins when the web application is initiated by a user.

2. Load Model
- The pre-trained ensemble model, which combines various machine learning algorithms with the final estimator being an Artificial Neural Network (ANN), is loaded into the web application.
- This model has been previously trained, evaluated, and saved as a `.pkl` file.

3. Get Data from User
- The web application prompts the user to input the necessary features required for making predictions.
- These features correspond to the original attributes used during the model training phase.

4. Preprocess Data
- The input data from the user is preprocessed to ensure it matches the format and scale of the data used during model training.
- This step may involve normalisation, encoding categorical variables, and handling missing values.

5. Make Prediction
- The preprocessed data is fed into the loaded ensemble model.
- The ensemble model processes the input data and generates a prediction based on the trained algorithms.

- The prediction could be a class label, a probability score, or any other relevant output.

6. Display Result
- The prediction result is then displayed to the user through the web application interface.
- The result could include additional information such as confidence scores or further recommendations.

This high-level deployment process outlines the steps from user interaction to final prediction display, ensuring that the web application is user-friendly and efficient in making accurate predictions using the ensemble model.

Diagram Explanation:

1. Start:
- Represents the initiation of the process when the user accesses the web application.

2. Load Model:
- The saved ensemble model is loaded into the application for use.

3. Get Data from User:
- The application collects input data from the user, which will be used for making predictions.

4. Preprocess Data:
- Input data is preprocessed to match the training data format, ensuring accurate predictions.

5. Make Prediction:
- The preprocessed data is passed to the ensemble model, which generates predictions.

6. Display Result:
- The application presents the prediction results to the user in an understandable format.

This diagram and explanation provide a clear and concise overview of the deployment process for the web application, ensuring users understand each step involved in generating predictions from their input data.

2.2 Logging

Logging is a crucial aspect of any web application, providing insights into the application's behaviour and helping in debugging, monitoring, and maintaining the system. For a machine learning web application, logging can capture details about data input, model predictions, errors, and system performance.

Key Components to Log:

1. User Inputs:

Log the data inputs provided by users. This helps in tracking the types of data being processed and can be useful for understanding user behaviour and improving the system.

2. Model Predictions:
   Log the predictions made by the model. This includes the output label or score, confidence levels, and any other relevant information.

3. Preprocessing Steps:
   Log the steps involved in preprocessing the data. This includes any transformations, normalizations, or encodings applied to the data before making predictions.

4. Errors and Exceptions:
   Log any errors or exceptions that occur during the execution of the application. This is critical for identifying and fixing issues in the system.

5. Performance Metrics:
   Log performance metrics such as response time, memory usage, and other system-level metrics. This helps in monitoring the application's performance and ensuring it runs efficiently.

Benefits of Logging:

1. Debugging:
   Logs provide detailed information about the application's execution, making it easier to identify and fix issues.

2. Monitoring:
   By logging performance metrics and user interactions, you can monitor the health and usage of the application in real-time.

3. Audit Trail:
   Logs create an audit trail of user inputs, model predictions, and system behaviour, which can be useful for compliance and analysis.

4. Improvement:
   Analysing logs can provide insights into how the application can be improved, such as identifying common user inputs that cause errors or slow performance.

2.3 Error Handling

Error handling is a critical aspect of any web application, ensuring that the system can gracefully manage unexpected situations without crashing or causing a poor user experience. In a machine learning web application, error handling is essential for dealing with various issues such as invalid user inputs, model prediction errors, and system-level failures.

Types of Errors to Handle:

1. User Input Errors:
- Errors that occur due to invalid or unexpected inputs from the user. This can include missing fields, incorrect data formats, and out-of-bound values.

2. Data Processing Errors:
- Errors that arise during data preprocessing steps such as transformations, normalizations, and encodings.

3. Model Prediction Errors:
- Errors that occur during the model prediction phase, including issues with loading the model, incompatible input data, and prediction failures.

4. System Errors:
- Errors related to the system infrastructure, such as file not found, network issues, memory overflows, and other runtime exceptions.

Benefits of Error Handling:

1. Improved User Experience:
- By providing meaningful error messages and feedback, users can understand what went wrong and how to correct it, leading to a better overall experience.

2. Increased Stability:
- Error handling prevents the application from crashing due to unexpected issues, ensuring continuous operation and reliability.

3. Enhanced Debugging:
- Logging errors helps developers quickly identify and resolve issues, reducing downtime and improving maintenance.

4. Security:
- Proper error handling prevents the application from exposing sensitive information, protecting against potential security threats.

Chapter 3: Performance

Importance of Performance Optimization:
Performance optimization is crucial for ensuring that your web application provides a fast, responsive, and seamless experience for users. For a machine learning web application, performance considerations include efficient data processing, quick model predictions, minimal latency, and optimal use of system resources. High performance also impacts user satisfaction, system scalability, and cost-efficiency.

Key Areas for Performance Optimization:

1. Data Processing Efficiency:
- Efficiently handling and processing large datasets is vital for performance. Techniques such as data batching, parallel processing, and efficient data structures can significantly reduce processing time.

2. Model Loading and Prediction Speed:
- Loading models into memory and making predictions quickly is essential for a responsive application. Techniques such as model serialisation, caching, and using optimised libraries can enhance performance.

3. Minimising Latency:
- Reducing latency involves optimising network requests, server response times, and minimising the amount of data transferred between the client and server.

4. Resource Management:
- Efficient use of CPU, memory, and disk resources ensures that the application can handle multiple requests simultaneously without performance degradation.

5. Scalability:
- Designing the application to scale horizontally (adding more servers) or vertically (upgrading server resources) ensures that it can handle increased user demand.

Performance Optimization Techniques:

1. Data Processing:
- Use efficient data structures (e.g., pandas DataFrame) for data manipulation.
- Implement parallel processing using libraries such as Dask or concurrent.futures.
- Use data batching to process data in chunks, reducing memory usage.

2. Model Optimization:
- Serialise models using efficient formats such as joblib or ONNX for quick loading.
- Use lightweight models or model quantization to reduce model size and inference time.
- Implement model caching to avoid reloading models for each request.

3. Code Optimization:
- Optimise algorithms and data structures for better performance.
- Profile and identify bottlenecks using tools like cProfile and line_profiler.
- Use just-in-time (JIT) compilation libraries like Numba to speed up Python code.

4. Server-Side Optimization:
- Use efficient web frameworks like FastAPI or Flask for handling requests.
- Implement asynchronous programming to handle multiple requests concurrently.
- Use a content delivery network (CDN) to serve static assets quickly.

5. Client-Side Optimization:
- Minimise the size of JavaScript and CSS files using minification and bundling.
- Optimise images and other media assets to reduce load times.
- Use lazy loading for non-essential resources to speed up initial page load.

6. Database Optimization:
- Index frequently queried fields to speed up database operations.
- Use database connection pooling to manage multiple connections efficiently.
- Optimise database queries to minimise response times.

Monitoring and Profiling:

1. Application Performance Monitoring (APM):
- Use APM tools such as New Relic, Datadog, or Dynatrace to monitor the application's performance in real-time.
- Track key metrics such as response times, error rates, and resource usage.

2. Profiling:
- Profile the application code to identify performance bottlenecks.
- Use tools like Py-Spy, line_profiler, and memory_profiler to profile CPU and memory usage.

3. Load Testing:
- Perform load testing using tools like JMeter, Locust, or Apache Bench to simulate high traffic and identify performance issues.
- Ensure the application can handle the expected user load with acceptable performance.

Conclusion

The high-level diagram illustrates the complete workflow of the machine learning web application, from data preprocessing to deployment and prediction. Initially, raw user data undergoes preprocessing to ensure consistency and quality. Pre-trained models, including Logistic Regression, K-Nearest Neighbor, Naive Bayes, Support Vector Machine, Decision Tree, and Artificial Neural Network, are loaded and combined into an ensemble model for robust predictions. Training and evaluation are performed using designated datasets to ensure model accuracy and generalizability. During deployment via Streamlit, the application processes user inputs, generates predictions using the ensemble model, and displays the results interactively. This streamlined approach ensures efficient data handling, accurate predictions, and user-friendly interaction, making the application both effective and scalable.