Architecture Document: Author Prediction System for Avila Bible:-

Content

---

1. Overview

This document provides a comprehensive architecture description for the Author Prediction System for the Avila Bible. The system predicts the authorship of text passages based on a preprocessed dataset and machine learning models. This document outlines the high-level architecture, component interactions, and deployment strategies.

2. Objectives

- Enable accurate prediction of text authorship based on predefined features.
- Provide a command-line or desktop interface for interacting with the system.
- Ensure scalability, reliability, and high availability for production deployment.

3. High-Level Architecture

The system consists of the following layers:

- **Data Layer**: Manages the dataset, preprocessing, and storage operations.
- **Model Layer**: Contains the trained machine learning models for author prediction.
- **Application Layer**: Handles user interactions and prediction workflows.
- **Deployment Layer**: Manages system hosting on a traditional server environment.

4. Component Descriptions

4.1 Data Loader Module

- Loads and splits the dataset into training and testing subsets.

4.2 Feature Extractor Module

- Extracts predefined features and target labels from the dataset.

4.3 Model Training Module

- Trains a supervised learning model using algorithms such as Random Forest or Gradient Boosting.

4.4 Model Prediction Module

- Uses the trained model to predict authorship for input features.

4.5 User Interface Module

- Provides a command-line or desktop interface for user interaction.

5. Interaction Diagram

- **User to User Interface Module**: Users input features and view predictions.
- **User Interface Module to Model Prediction Module**: Processes input features and generates predictions.

6. Data Flow Diagram

Level 1:

1. User submits input features via the user interface.
2. Input features are passed to the prediction module.
3. The model generates predictions and returns them to the user interface.

7. Deployment Diagram

- **Platform**: Hosted on Render or a traditional server environment.
- **Components**:
    - User interface executable or script.
    - Backend processing modules for model inference.
    - Storage for logs and dataset management.
- **Communication**: Direct execution of processes.

8. Technology Stack

- **Programming Languages**: Python (Backend)
- **Frameworks**: Flask (direct script execution)
- **Database**: None (local file storage for logs and models)
- **Machine Learning Libraries**: scikit-learn
- **Deployment Tools**: Simple deployment on a traditional server or desktop.

9. Non-Functional Requirements

- **Scalability**: Handle increased user inputs by improving hardware resources.
- **Reliability**: Ensure consistent performance during executions.
- **Security**: Validate user inputs to avoid errors.
- **Performance**: Respond to prediction requests within 200 ms.

10. Scalability and Performance Considerations

- Optimize model inference by loading models into memory during execution.
- Utilize batch processing for multiple predictions if necessary.