Low-Level Design (LLD) for Red Wine Quality Prediction

1. Introduction

The Low-Level Design (LLD) provides a detailed technical overview of the components outlined in the High-Level Design (HLD). This document includes the actual data flow, algorithms, database schema, API design, and configuration settings for the system.

2. Data Source and Preprocessing

2.1 Data Format

- The dataset consists of a CSV file containing physicochemical properties (such as pH, alcohol content, acidity) and a column for wine quality.
- Input data schema:
    - `fixed_acidity`: float
    - `volatile_acidity`: float
    - `citric_acid`: float
    - `residual_sugar`: float
    - `chlorides`: float
    - `free_sulfur_dioxide`: float
    - `total_sulfur_dioxide`: float
    - `density`: float
    - `pH`: float
    - `sulphates`: float
    - `alcohol`: float
    - `quality`: integer (0–10 scale)

2.2 Preprocessing Steps

- **Missing Value Handling:** Use median imputation for continuous variables if any values are missing.
- **Outlier Detection:** Implement Z-score or IQR (Interquartile Range) to detect and remove extreme outliers.
- **Feature Scaling:** Use StandardScaler to scale features like acidity, pH, alcohol, and others.
- **Train-Test Split:** Split the data into 80% training and 20% testing.
- **Data Augmentation (Optional):** Techniques like SMOTE (Synthetic Minority Oversampling Technique) for handling class imbalance.

3. Model Selection

3.1 Algorithms

- **Algorithm 1: Random Forest**
  - Number of trees: 100
  - Max depth: 10 (to avoid overfitting)
  - Criterion: Gini or Entropy
- **Algorithm 2: Support Vector Machines (SVM)**
  - Kernel: Radial Basis Function (RBF)
  - Regularization (C): 1.0
- **Algorithm 3: Gradient Boosting**
  - Number of boosting stages: 100
  - Learning rate: 0.1

3.2 Model Training

- **Cross-Validation:** 5-fold cross-validation for model evaluation.
- **Hyperparameter Tuning:** Use GridSearchCV to optimize model parameters (e.g., depth of trees, learning rate for Gradient Boosting).
- **Evaluation Metrics:**
  - **Accuracy**
  - **Precision, Recall, F1-Score**
  - **Confusion Matrix**
  - **ROC-AUC Score**

4. Database Design

4.1 Database Schema

- **Table 1: Wine Quality Predictions**
  - `id` (Primary Key)
  - `wine_features` (JSON or separate columns for each physicochemical property)
  - `predicted_quality` (Integer)
  - `confidence_level` (Float)
  - `timestamp` (Datetime)
- **Table 2: Model Metadata**
  - `model_id` (Primary Key)
  - `model_type` (e.g., RandomForest, SVM)
  - `parameters` (JSON format)
  - `accuracy` (Float)
  - `date_trained` (Datetime)

4.2 Database Technology

- **SQLite** (for local development) or **PostgreSQL** (for production)
- Index columns such as `predicted_quality` and `timestamp` for faster queries.

5. API Design

5.1 Endpoints

- **POST /predict**
    - **Input:** JSON payload with wine features (e.g., acidity, pH, etc.)
    - **Output:** Predicted wine quality score and confidence interval.

Example Payload:
json


```json
{
    "fixed_acidity": 7.4,
    "volatile_acidity": 0.7,
    "citric_acid": 0.0,
    "residual_sugar": 1.9,
    "chlorides": 0.076,
    "free_sulfur_dioxide": 11.0,
    "total_sulfur_dioxide": 34.0,
    "density": 0.9978,
    "pH": 3.51,
    "sulphates": 0.56,
    "alcohol": 9.4
}
```


- **GET /model-info**
    - **Output:** Returns metadata of the deployed model (e.g., algorithm used, accuracy, training date).

Response Example:
json


```json
{
    "model_type": "RandomForest",
    "accuracy": 0.85,
    "trained_on": "2024-01-01"
}
```


6. System Workflow

1. **User Input:** A user submits physicochemical data through the web interface.
2. **Backend Processing:**

    - The backend API triggers data preprocessing steps (scaling, encoding, etc.).
    - The machine learning model is loaded to predict wine quality.
3. **Prediction Output:**
    - The predicted wine quality is returned along with a confidence score.
    - Optionally, a breakdown of feature importance can be shown to explain why a particular score was predicted.
4. **Data Storage:**
    - The input data and prediction are stored in the database for future analysis.

7. Frontend Design

7.1 User Interface (UI)

- **Form Input:** Users can manually enter the wine characteristics or upload a CSV file.
- **Prediction Display:** Display the predicted wine quality score with a confidence level.
- **Historical Data View:** Allow users to view past predictions with timestamps.

7.2 Technologies

- **HTML/CSS/JavaScript:** For the frontend interface.
- **Frameworks:** React.js or Vue.js for a dynamic interface.

8. Deployment

8.1 Containerization

- **Docker**: A Dockerfile will be created for the application, enabling easy containerization and deployment on various platforms.

8.2 Cloud Deployment

- **Amazon Web Services (AWS) EC2**: The application can be hosted on an EC2 instance.
- **AWS S3**: Can be used for storing large datasets.
- **Elastic Beanstalk**: To manage deployments of the entire application.

8.3 Version Control

- **Git**: The source code will be maintained in a Git repository with proper versioning.

9. Security Considerations

- **Authentication**: Implement user authentication using JWT (JSON Web Tokens) for secure access to the API.
- **Data Encryption**: Ensure that sensitive user inputs are encrypted, both in transit (using HTTPS) and at rest (in the database).

10. Testing Strategy

10.1 Unit Testing

- Test each individual component (e.g., data preprocessing, prediction).

10.2 Integration Testing

- Ensure that the entire system works seamlessly, from data input to prediction.

10.3 Load Testing

- Test the API under different loads to ensure scalability.