**Equipment Failure Prediction Application Documentation**

**Introduction**

This documentation provides a detailed guide for the **Equipment Failure Prediction Application** built using Flask for the backend and a machine learning model pipeline that predicts equipment failure based on various features such as Age, Usage Hours, and environmental factors. The application utilizes a Voting Classifier comprising multiple machine learning models to ensure robust predictions.

**Application Overview**

The Equipment Failure Prediction Application predicts whether equipment will fail based on the following features:

***Numerical features:***

 Age

 Usage Hours

 Maintenance History

 Temperature

 Pressure

 Vibration Level

 Operator Experience

 Failure History

***Categorical features:***

 Location

 Environment

The application is structured to handle HTTP requests from a frontend (React.js) that sends data to the backend Flask server for prediction. The server processes the data, runs it through the machine learning pipeline, and returns the prediction.

**Flask Backend Setup:**

The Flask backend serves as the core of the application. It manages incoming HTTP requests, processes data, and returns the results. **CORS** is enabled to allow requests from the frontend, ensuring secure communication.

**Key Packages**

* **Flask**: Micro web framework to handle routing and HTTP requests.
* **Flask-CORS**: Enables Cross-Origin Resource Sharing (CORS) to allow the frontend to interact with the backend.
* **Numpy** & **Pandas**: Used for data manipulation and numerical operations.
* **Scikit-learn**: Includes essential preprocessing steps and machine learning models.

**Flask Routes:**

* **/predict**: Handles POST requests for prediction.
* **/**: Handles the rendering of the HTML template for the home page.

**Data Preprocessing:**

***Dataset:***

The dataset used for this model is read from a CSV file named equipment\_failure\_dataset.csv. The data includes both numerical and categorical features essential for predicting equipment failure. The column Failure is the target variable, where 'Yes' indicates failure, and 'No' indicates non-failure.

***Preprocessing Pipelines:***

Two preprocessing pipelines are defined: one for numerical features and another for categorical features.

* **Numerical Pipeline**:
  + Missing data is imputed using the mean of the column values.
  + The numerical values are scaled to a standard range using **StandardScaler** to ensure the models do not give undue importance to features with larger scales.
* **Categorical Pipeline**:
  + **OneHotEncoder** is used to encode categorical features. This encoder handles unseen categories gracefully during prediction by ignoring them.

These pipelines are then combined using the ColumnTransformer to apply transformations to their respective columns.

**Model Building:**

The backend uses a **Voting Classifier** that aggregates predictions from several individual models to make the final prediction. This improves accuracy and robustness by combining the strengths of multiple algorithms.

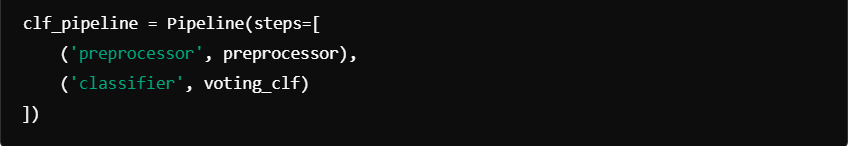
**Models Used:**

1. **RandomForestClassifier**
2. **GradientBoostingClassifier**
3. **LogisticRegression**
4. **Support Vector Classifier (SVC)** with probability estimates enabled
5. **KNeighborsClassifier**
6. **DecisionTreeClassifier**
7. **XGBoostClassifier** (requires xgboost package)

These models are wrapped into a **VotingClassifier** with soft voting, which means the final prediction is based on the probability estimates from each model.

***Model Training:***

The model is trained on 70% of the data, and the remaining 30% is reserved for testing. The entire training process, including preprocessing and model building, is integrated into a **scikit-learn pipeline**.



**Prediction Endpoint:**

The prediction functionality is exposed through the **/predict** endpoint. This endpoint expects a POST request containing JSON data with the required features. Once the data is validated, it is transformed into a DataFrame and passed through the trained model pipeline to generate a prediction.

**Input Validation:**

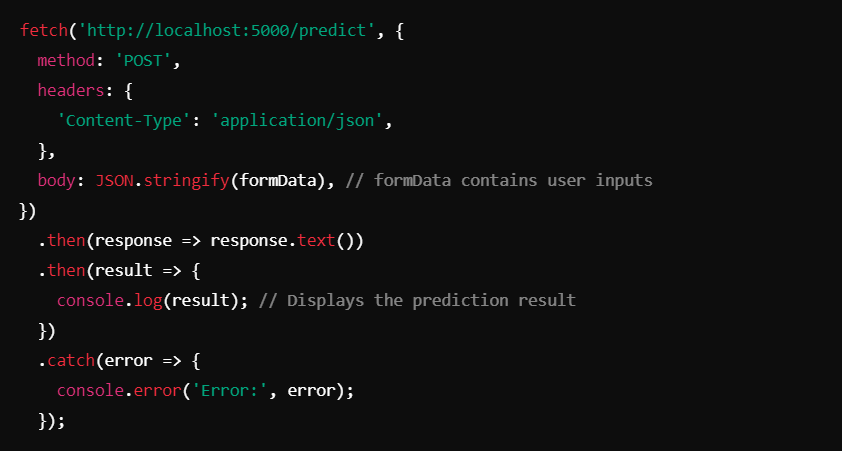
* The application checks for the presence of all necessary features.
* If any required field is missing or improperly formatted, an error is returned with a 400 status code.
* The features are cast into the appropriate data types (e.g., float for numerical features) to ensure compatibility with the model.

**Response:**

* If the prediction is successful, the result is returned in plain text as either Equipment fails or Equipment not fails, with a 200 status code.
* In case of failure during prediction (e.g., if the model encounters an error), an error message is logged, and a 500 status code is returned.

**Frontend Integration:**

The backend supports requests from a **React.js frontend**. **CORS** is configured to allow requests only from http://localhost:3000, ensuring security between the frontend and backend.



**Conclusion:**

This Equipment Failure Prediction Application is built for high accuracy and robustness, leveraging an ensemble of machine learning models for prediction. It is designed to be easily integrated with a frontend application using modern technologies like React.js, and its backend is capable of handling real-world equipment failure scenarios with flexibility and reliability.