Combined Cycle Power Plant Performance Prediction

Introduction

A Combined Cycle Power Plant (CCPP) integrates both gas and steam turbines to generate electricity efficiently. The waste heat from the gas turbine is used to power a steam turbine, improving overall energy conversion. To optimize performance, it is essential to predict power output based on operational parameters.

Machine Learning (ML) techniques such as Linear Regression, Decision Trees, and Random Forests can analyze plant conditions and accurately predict power output. This project focuses on developing an ML-based system to estimate power generation using key features like Ambient Temperature (AT), Exhaust Vacuum (V), Ambient Pressure (AP), and Relative Humidity (RH).

Objectives

By the end of this project, you will:

- Understand the impact of operational parameters on CCPP performance.

- Apply Machine Learning techniques to build predictive models.

- Train and evaluate models using performance metrics.

- Develop a Flask web application to provide real-time predictions.

- Optimize and compare different ML models to enhance accuracy.

Project Workflow

1. Data Collection: Use publicly available datasets on CCPP performance.

1. Data Preprocessing: Handle missing values, normalize data, and split into training and testing sets.

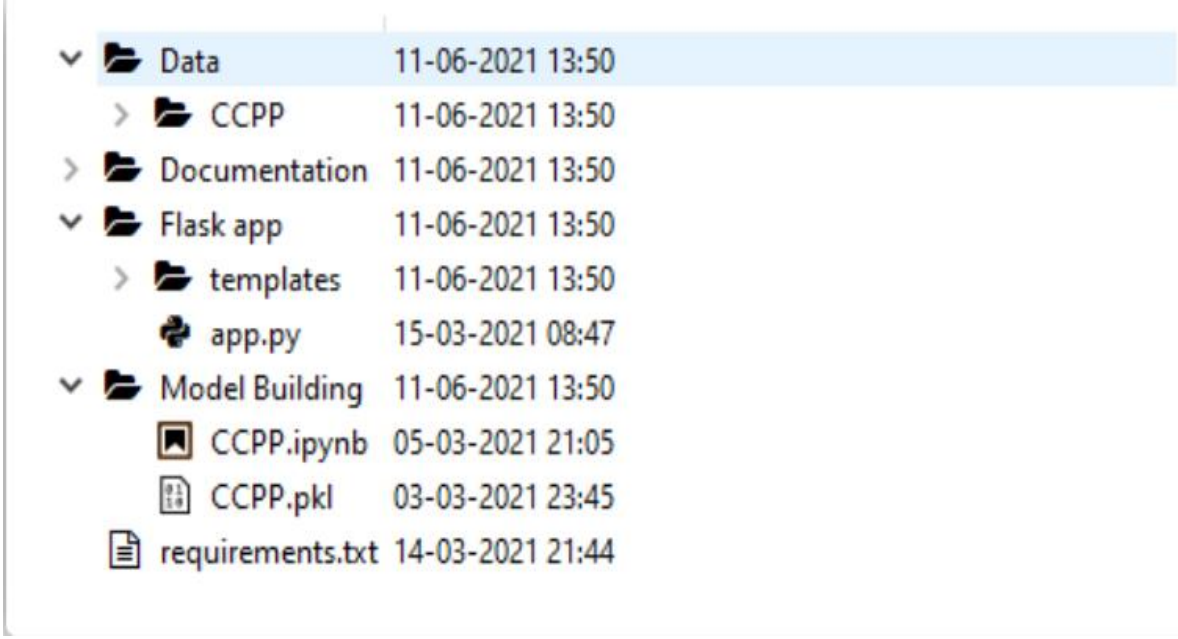
1. Model Building: Train Linear Regression, Decision Tree, and Random Forest models.

1. Model Evaluation: Measure accuracy using metrics like RMSE and R-squared.

1. Application Development: Deploy the model via a Flask web app.

1. Deployment: Integrate a user-friendly web interface for predictions.

Project Structure



Technical Architecture

- Frontend: HTML, CSS, JavaScript for user interaction.

- Backend: Flask handles HTTP requests and serves predictions.

- Machine Learning Models: Implemented using scikit-learn.

- Database: CSV dataset for training and testing models.

Data Collection

- **Dataset Source:** The dataset is collected from publicly available sources and contains operational data from Combined Cycle Power Plants.
- **Dataset Features:**
    - **AT** (Ambient Temperature in °C)
    - **V** (Exhaust Vacuum in cm Hg)
    - **AP** (Ambient Pressure in millibar)
    - **RH** (Relative Humidity in %)
    - **PE** (Power Output in MW - target variable)
- **Dataset Structure:**
    - The dataset consists of multiple instances of power plant operating conditions, with each record representing a specific timestamp.
    - The data is stored in CSV format and split into **training (80%)** and **testing (20%)** sets.

Data Preprocessing

Handling Missing Values:

Check for missing values and apply imputation techniques if necessary.

Normalization:

Scale features using Min-Max scaling or Standardization to improve model performance.

Feature Engineering:

Analyze feature correlations to remove irrelevant or redundant data.

Generate new features if needed to enhance model prediction accuracy.

Data Splitting:

The dataset is divided into training (80%) and testing (20%) sets to evaluate model performance.

Model Building

- Linear Regression: Predicts PE using a linear relationship with input features.

- Decision Tree: Splits data into decision nodes based on feature importance.

- Random Forest: An ensemble method using multiple decision trees to reduce overfitting.

Model Evaluation

Metrics used for evaluation:
- RMSE (Root Mean Squared Error): Measures prediction error.
- R^2 Score: Indicates model fit.
- Comparison: Random Forest is expected to perform better due to its ensemble approach.

Model Training

Training Process;

1. **Load the dataset:** Read the XLSX file using pandas.
2. **Preprocess the data:** Handle missing values, normalize features, and split into training and testing sets.
3. **Train different models:** Implement Linear Regression, Decision Tree, and Random Forest models.
4. **Optimize hyperparameters:** Use GridSearchCV or RandomizedSearchCV to fine-tune model parameters.
5. **Save the best model:** Store the trained model as CCPP.pkl using pickle.

Training Script (train_model)

```
# To handle data in form of rows and columns
import pandas as pd

# Numerical libraries
import numpy as np

#importing ploting libraries
import matplotlib.pyplot as plt

#importing seaborn for statistical plots
import seaborn as sns

#implements serialization
import pickle
```

```
from google.colab import files
files.upload()
```

```
data = pd.read_excel("Folds5x2_pp.ods", header=0 , names = ['AT','V','AP','RH','PE'])
```

```
print(data.isnull().sum())
print(data.head())
print(data.describe().T)
```

Show hidden output

```
plt.scatter(data['AT'],data['PE'])
plt.scatter(data['V'],data['PE'])
plt.scatter(data['AP'],data['PE'])
plt.scatter(data['RH'],data['PE'])
sns.pairplot(data,diag_kind = 'hist')
```

Show hidden output

```
x = data.drop(['PE'],axis=1)
y = data['PE']
```

```
from sklearn.model_selection import train_test_split
x_train, xtest, y_train, ytest = train_test_split(x, y, test_size = 0.2, random_state = 0)
x_train.shape
print(x_train.shape)
xtest.shape
print(xtest.shape)
```

```
(7654, 4)
(1914, 4)
```

```
# Import the machine Learning models
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
# Initializing the models
LRmodel=LinearRegression()
DTmodel=DecisionTreeRegressor()
RFmodel=RandomForestRegressor()
from sklearn.linear_model import LinearRegression
LRmodel = LinearRegression()
LRmodel.fit(x_train, y_train)
print(LRmodel.fit(x_train, y_train))
from sklearn.tree import DecisionTreeRegressor
DTRmodel = DecisionTreeRegressor()
DTRmodel.fit(x_train, y_train)
print(DTRmodel.fit(x_train, y_train))
from sklearn.ensemble import RandomForestRegressor
RFmodel = DecisionTreeRegressor()
RFmodel.fit(x_train, y_train)
print(RFmodel.fit(x_train, y_train))
```

```
# Linear Regression
from sklearn.linear_model import LinearRegression
# Initializing the model
LRmodel = LinearRegression()
# Train the data with Linear Regreesion model
LRmodel.fit(x_train, y_train)
LinearRegression()
LRpred=LRmodel.predict(xtest)
# Importing R Square Library
from sklearn.metrics import r2_score
# Checking for accuracy score with actual data and predicted data
LRscore=r2_score(ytest, LRpred)
LRscore
print(LRscore)
```

```
0.932531554761302
```

```
# Decision Tree Regression
from sklearn.tree import DecisionTreeRegressor
# Initializing the model
DTRmodel = DecisionTreeRegressor()
# Train the data with Linear Regreesion model
DTRmodel.fit(x_train, y_train)
DecisionTreeRegressor()
DTRpred=DTRmodel.predict(xtest)
# Importing R Square Library
DTRscore=r2_score(ytest, DTRpred)
DTRscore
print(DTRscore)
```

```python
# Random Forest Regressor
from sklearn.ensemble import RandomForestRegressor
# Initializing the model
RFmodel = RandomForestRegressor()
RFmodel.fit(x_train, y_train)
RandomForestRegressor()
RFpred=RFmodel.predict(xtest)
# Importing R Square Library
RFscore=r2_score(ytest, RFpred)
RFscore
print(RFscore)
```

```
0.9652471732590461
```

```python
pickle.dump(RFmodel, open('CCPP.pkl','wb'))
```

Flask Web Application

Backend (app.py)

- Handles HTTP requests and serves predictions.
- Loads the trained model (CCPP.pkl).
- Accepts user input (AT, V, AP, RH) via an API.
- Returns predicted **power output (PE)** as JSON.

app.py > ...

```python
from flask import Flask, request, jsonify, render_template
from flask_cors import CORS
import pickle
import numpy as np

# Load the saved model
model = pickle.load(open('CCPP.pkl', 'rb'))

app = Flask(__name__)
CORS(app)  # Enable Cross-Origin Resource Sharing

@app.route('/')
def home():
    return render_template('home.html')  # Serve home.html as the first page

@app.route('/index1')
def index1():
    return render_template('index1.html')  # Serve index1.html when requested

@app.route('/predict', methods=['POST'])
def predict():
    data = request.json  # Get JSON data from frontend

    # Extract input features
    AT = float(data['AT'])
    V = float(data['V'])
    AP = float(data['AP'])
    RH = float(data['RH'])

    # Make prediction
    input_data = np.array([[AT, V, AP, RH]])
    prediction = model.predict(input_data)

    return jsonify({'Predicted Power Output': prediction[0]})

if __name__ == '__main__':
    app.run(port=5000, debug=True)
```

Frontend (index1.html & home.html)

Users enter operational parameters.

The system predicts power output in real-time.

HOME.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Combined Cycle Power Plant</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 0;
            padding: 0;
            background-color: #f8f8f8;
        }
        .header {
            background-color: #6a0dad;
            color: white;
            padding: 15px;
            font-size: 20px;
            font-weight: bold;
        }
        .header a {
            color: white;
            text-decoration: none;
            margin-left: 20px;
            float: right;
        }
        .container {
            background-color: #f4c842;
            padding: 20px;
            margin: 20px;
            border-radius: 8px;
        }
        h2 {
            color: #333;
        }
        p {
            color: #333;
            font-size: 16px;
            line-height: 1.5;
        }
    </style>
</head>
<body>
    <div class="header">
        COMBINED CYCLE POWER PLANT
        <a href="{{ url_for('index1') }}">Go to Predict</a>
        <a href="#">Home</a>
    </div>
    <div class="container">
        <h2>Introduction</h2>
        <p>The Combined Cycle Power Plant or combined cycle gas turbine, a gas turbine generator generates electricity and waste heat is used to make steam to generate additional electricity via a steam turbine...</p>
    </div>
</body>
</html>
```

INDEX.HTML

```
<!DOCTYPE html>
<html>
<head>
    <title>Prediction Page</title>
    <style>
        .header{
            text-align: center;
            background-color: blue;
            color: white;
            padding: 20px;
            margin: 10px 70px;
            border-radius: 30px;
        }
        .container{
            text-align: center;
        }
    </style>
</head>
<body style="display: flex;align-items: center;justify-content: center;">
    <div style="width: 70%;background-color: rgb(246, 246, 138);">
        <h2 style="text-align: center;">Power Plant Output Prediction</h2>
        <div class="container">
            <div class="header">PREDICTION OF ELECTRICAL OUTPUT POWER OF COMBINED CYCLE POWER PLANT</div>
            <form id="prediction-form">
                <label for="AT">Ambient Temperature (AT):</label><br>
                <input type="text" id="AT" name="AT"><br><br>

                <label for="V">Exhaust Vacuum (V):</label><br>
                <input type="text" id="V" name="V"><br><br>

                <label for="AP">Ambient Pressure (AP):</label><br>
                <input type="text" id="AP" name="AP"><br><br>

                <label for="RH">Relative Humidity (RH):</label><br>
                <input type="text" id="RH" name="RH"><br><br>

                <button type="button" onclick="predict()">Predict</button>
            </form>
            <h3>Predicted Output: <span id="result"></span></h3>
        </div>
    </div>
</body>
</html>
```

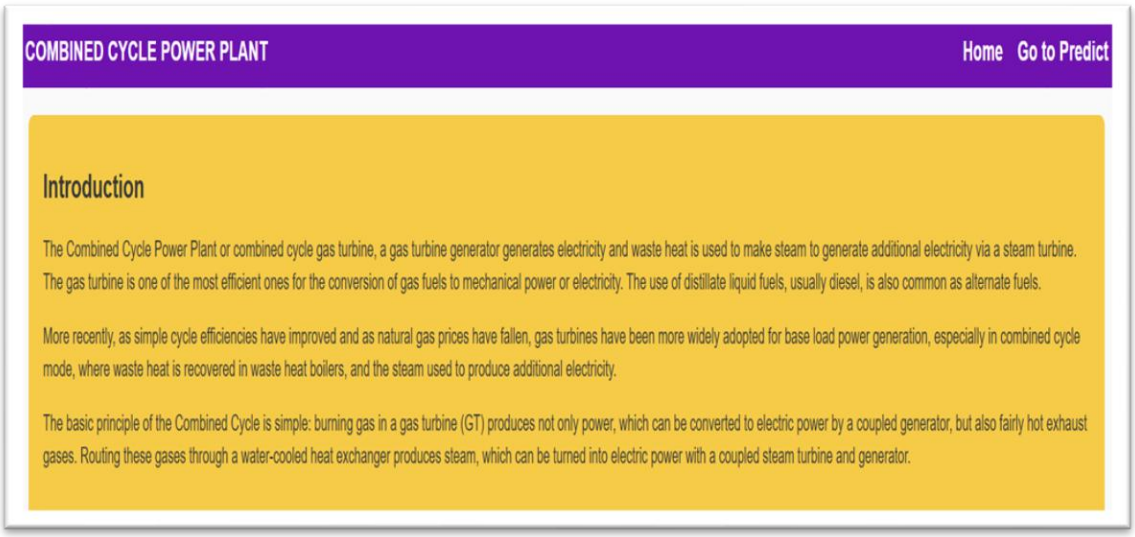
Running the Code

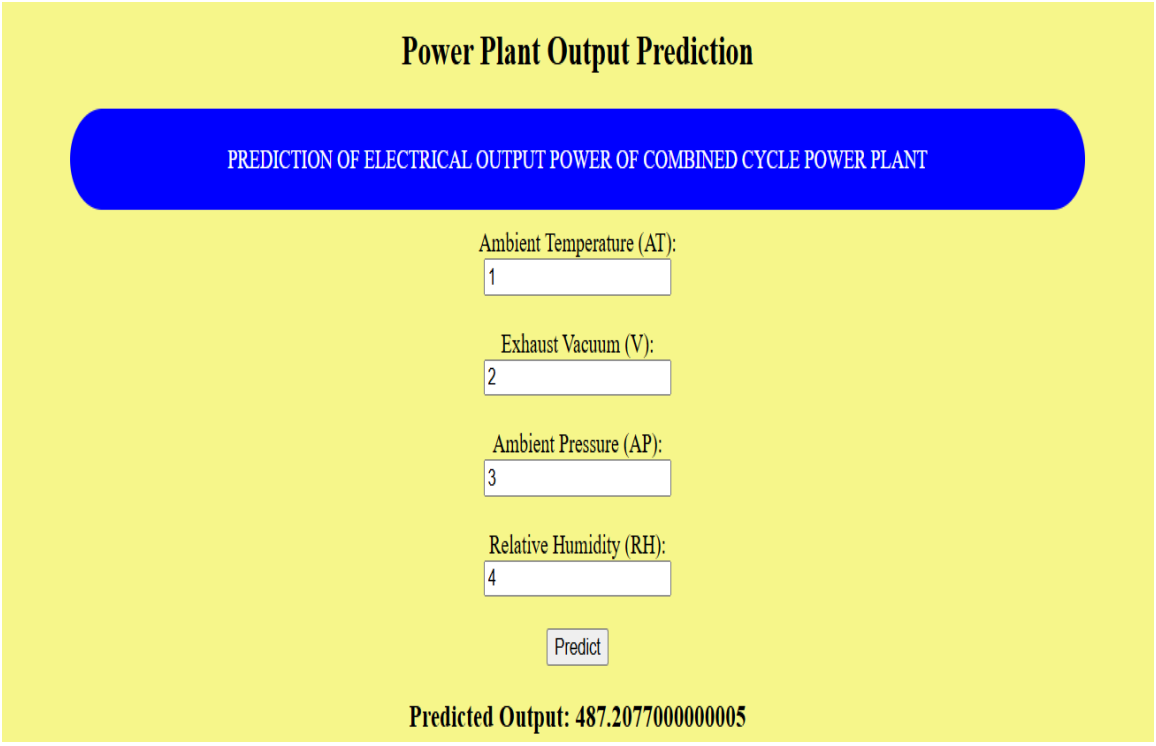
The execution of the project begins with training the deep learning model and then deploying it via a Flask web application. The process starts by running the train_model.py script, which loads the dataset, preprocesses the images, and trains a convolutional neural network using transfer learning techniques . Once training is completed, the model is saved as CCPP.pkl, which serves as the core of the prediction system.

After the model is trained and saved, app.py is executed to launch the Flask web server. The Flask application initializes by loading CCPP.pkl, setting up routes, and rendering HTML templates. When a user accesses the web application through http://127.0.0.1:5000/, they are directed to the homepage, index.html.

```
PS C:\Users\gandi teja\OneDrive\Desktop\FLASK> & "C:/Users/gandi teja/AppData/Local/Programs/Python/Python313/python.exe" "c:/Users/gandi teja/OneDrive/Desktop/FLASK/
app.py"
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 297-949-952
```

From the homepage (home.html), users can navigate to the prediction page (index1.html), where they can enter power plant operational parameters (AT, V, AP, RH). When the user submits the form, the input values are sent to the Flask backend, which processes the data using the trained model and predicts the power output. The result is then displayed on the same page. Users can enter new values for additional predictions or navigate back to the homepage. This setup ensures seamless interaction between the backend and frontend components, making the system user-friendly and efficient.


COMBINED CYCLE POWER PLANT
Home
Go to Predict
Introduction
The Combined Cycle Power Plant or combined cycle gas turbine, a gas turbine generator generates electricity and waste heat is used to make steam to generate additional electricity via a steam turbine. The gas turbine is one of the most efficient ones for the conversion of gas fuels to mechanical power or electricity. The use of distillate liquid fuels, usually diesel, is also common as alternate fuels.
More recently, as simple cycle efficiencies have improved and as natural gas prices have fallen, gas turbines have been more widely adopted for base load power generation, especially in combined cycle mode, where waste heat is recovered in waste heat boilers, and the steam used to produce additional electricity.
The basic principle of the Combined Cycle is simple: burning gas in a gas turbine (GT) produces not only power, which can be converted to electric power by a coupled generator, but also fairly hot exhaust gases. Routing these gases through a water-cooled heat exchanger produces steam, which can be turned into electric power with a coupled steam turbine and generator.


Power Plant Output Prediction
PREDICTION OF ELECTRICAL OUTPUT POWER OF COMBINED CYCLE POWER PLANT
Ambient Temperature (AT):
1
Exhaust Vacuum (V):
2
Ambient Pressure (AP):
3
Relative Humidity (RH):
4
Predict
Predicted Output: 487.2077000000005

Conclusion

This project demonstrates the potential of Machine Learning in power plant performance optimization. The trained model enables power plants to predict output efficiently, aiding in decision-making and improving energy utilization.

Future Enhancements:
- Incorporate real-time sensor data.
- Implement deep learning models for enhanced accuracy.
- Deploy on a cloud platform for scalability.

By leveraging ML and Flask, this system provides a practical approach to improving Combined Cycle Power Plant efficiency. 🚀