FOREST FIRE WEATHER INDEX (FWI) PREDICTION USING MACHINE LEARNING

Infosys Springboard Internship Project

Submitted by

Namasani Surya Teja

Academic Year

2024 – 2025

Contents

SECTION 1

DATASET AND PREPROCESSING

1.1 Overview of the Dataset

You use a forest fire dataset created to study the relationship between weather conditions, fire weather indices, and fire occurrence. Each record corresponds to observations collected on a specific day. The dataset supports predictive modeling where the main objective is to classify whether a fire occurs or not based on environmental factors.

The dataset is stored in CSV format, which is widely used for structured data storage. It is loaded into Python using the Pandas library, which provides efficient tools for data inspection, transformation, and preprocessing. Since the dataset contains both numerical and categorical attributes, preprocessing is mandatory before applying any machine learning algorithm.

1.2 Dataset Attributes and Their Significance

- The dataset consists of temporal variables, meteorological measurements, fire indices, and a categorical output variable.
- The day attribute represents the calendar day of observation. Daily variations are important because weather conditions change rapidly and directly affect fire risk.
- The month attribute captures seasonal variation. Fire incidents are more frequent in dry seasons, making this attribute highly relevant.
- The year attribute provides long-term temporal context and helps in identifying trends across years.
- Temperature is measured in degrees Celsius and directly influences vegetation dryness. Higher temperatures increase the likelihood of ignition.
- RH, or Relative Humidity, represents moisture content in the air. Lower humidity levels contribute to dry fuel conditions.
- Ws, or Wind Speed, measured in kilometers per hour, plays a major role in fire spread by increasing oxygen supply and flame movement.
- Rain represents rainfall in millimeters. Rain reduces fuel dryness and lowers fire probability.
- FFMC indicates moisture content of fine fuels like leaves and twigs. It is sensitive to short-term weather changes.
- DMC represents moisture in moderately deep organic layers and reflects medium-term drying.
- DC represents long-term drought effects by measuring moisture in deep compact layers.
- ISI estimates the expected rate of fire spread immediately after ignition.
- BUI represents the total amount of combustible material available.
- FWI combines several indices to estimate overall fire intensity.
- Classes is the output variable. It contains two labels, fire and not fire, making this a binary classification problem.

1.3 Dataset Loading

The dataset is loaded using the Pandas read_csv function. This converts the CSV file into a DataFrame, which allows easy access to rows and columns.

Printing the dataset after loading helps verify successful file import, correct column names, and proper alignment of values. This step ensures no structural issues exist before preprocessing begins.

1.4 Data Preprocessing Code

The following Python code is used for dataset loading and preprocessing:

```
import pandas as pd
import numpy as np

# Load dataset
df = pd.read_csv("dataset.csv")
print(df)

# Check for missing values
print(df.isna().sum())

# Replace missing values with 0
df = df.fillna(0)

# Convert categorical columns to numerical form
for col in df.select_dtypes(include=['object']).columns:
    df[col] = df[col].astype('category').cat.codes

# Display processed data
print(df.head())
```

1.5 Missing Value Detection

- Missing values are common in real-world datasets due to sensor errors, data collection issues, or manual entry mistakes. These missing values are represented as NaN in Pandas.
- You identify missing values using the isna().sum() function, which counts missing entries in each column. This step is essential because machine learning models cannot handle NaN values directly.
- Detecting missing values early prevents training failures and unreliable predictions.

1.6 Missing Value Handling

- After identifying missing values, you replace them with zero using the fillna(0) method.

- This approach ensures that no rows are removed from the dataset, preserving the full data volume. Zero imputation provides numerical consistency and allows the model to process all records without errors.
- While zero may not always reflect real-world conditions, it is acceptable for baseline models and ensures smooth training.

1.7 Identification of Categorical Variables

- Machine learning algorithms require numerical inputs. Therefore, categorical variables must be detected and converted.
- You identify categorical columns by selecting columns with the object data type. This ensures that all text-based columns are handled appropriately before modeling.
- In this dataset, the Classes column is categorical and represents the target variable.

1.8 Encoding of Categorical Data

Categorical values are converted into numerical codes using category encoding.

Each unique label is mapped to an integer value. For example:

- not fire is encoded as 0
- fire is encoded as 1

This transformation preserves class distinction while making the dataset compatible with machine learning algorithms.

1.9 Final Data Validation

After preprocessing, the first few rows of the dataset are displayed using the head() function.

This confirms that:

- All missing values are handled
- All columns are numeric
- The dataset is ready for model training

Final validation ensures data quality and consistency.

1.10 Importance of Data Preprocessing

- Data preprocessing improves model performance, stability, and accuracy. Clean and well-prepared data reduces noise, prevents errors, and ensures meaningful learning.
- This preprocessing step fulfills all dataset preparation requirements and prepares the data for exploratory analysis and model development.

SECTION 2
DATASET STATISTICS

2.1 Purpose of Dataset Statistics

Dataset statistics help you understand the internal structure, distribution, and relationships between variables before model building. Statistical analysis reveals patterns, dependencies, and variability in data. This step reduces guesswork and supports informed model selection.

In this section, you analyze statistical relationships using correlation analysis and feature distributions using histograms. These methods provide numerical and visual insight into how features interact and how values are spread.

2.2 Preparation for Statistical Analysis

Before performing statistical analysis, the dataset is prepared to ensure consistency and correctness. Missing values are replaced with zeros, and categorical variables are converted into numerical form. This ensures that correlation calculations and visualizations run without errors.

The dataset is then filtered to include only relevant numerical features related to weather and fire indices. These features directly influence fire behavior and are essential for statistical interpretation.

2.3 Statistical Analysis Code

The following Python code is used for dataset statistics, correlation analysis, and visualization.

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns


df = pd.read_csv("dataset.csv")


# Handle missing values

df = df.fillna(0)


# Encode categorical columns

for col in df.select_dtypes(include=["object"]).columns:

    df[col] = df[col].astype("category").cat.codes
```

```
# Select numerical columns for correlation
cols = ["Temperature","RH","Ws","Rain","FFMC","DMC","DC","ISI","BUI","FWI"]

# Correlation matrix
corr = df[cols].corr()
print(corr)

# Correlation heatmap
plt.figure(figsize=(8,6))
sns.heatmap(corr, annot=True, fmt=".2f", cmap="coolwarm", linewidths=0.5)
plt.title("Correlation Heatmap with Values")
plt.tight_layout()
plt.show()
```

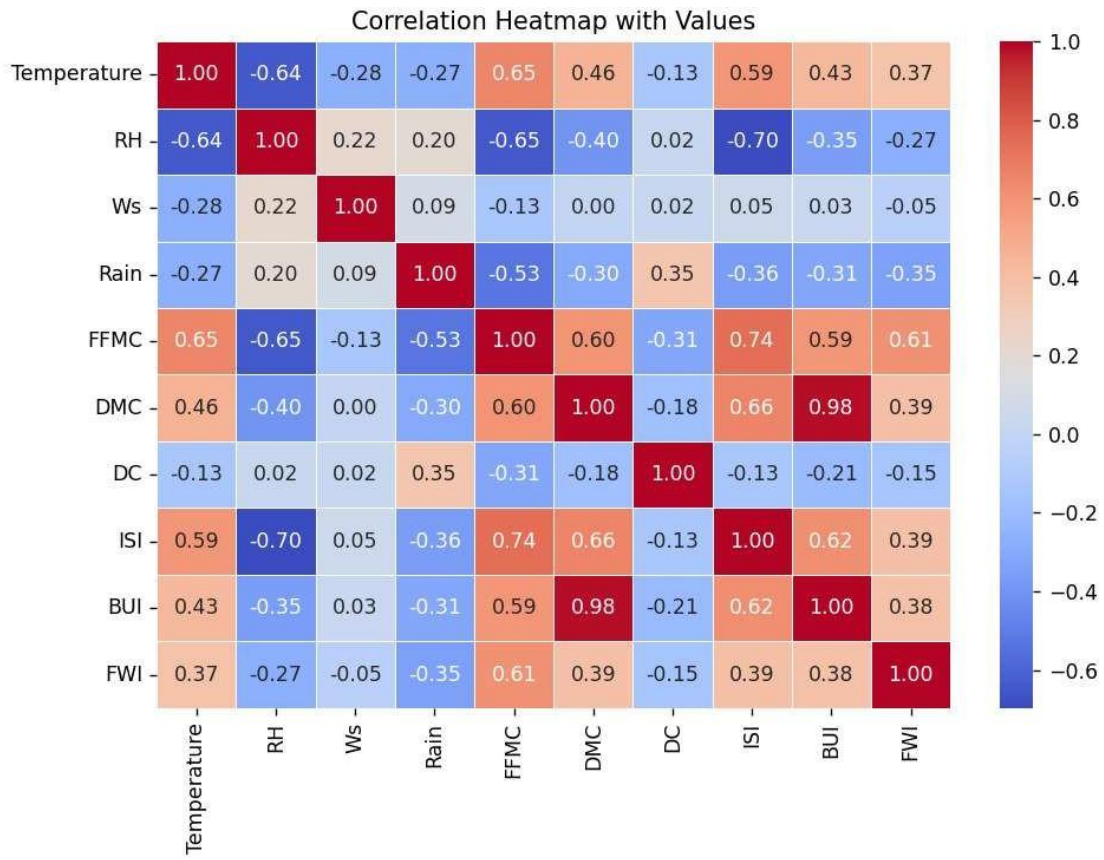
2.4 Correlation Analysis

Correlation analysis measures the strength and direction of relationships between numerical variables. The correlation coefficient ranges from -1 to +1.

- A value close to +1 indicates strong positive correlation.
- A value close to -1 indicates strong negative correlation.
- A value near 0 indicates weak or no correlation.
- The printed correlation matrix provides numerical values for every feature pair. This matrix helps identify which variables influence each other most strongly.
- Fire-related indices such as FFMC, ISI, BUI, and FWI show strong positive correlations. This confirms that these indices are mathematically and conceptually related. Temperature shows moderate positive correlation with FFMC and FWI, indicating that higher temperatures increase fire risk. Relative Humidity shows negative correlation with most fire indices, meaning higher humidity reduces fire intensity.
- Rain shows weak or negative correlation with fire indices, which aligns with real-world behavior where rainfall suppresses fires.

2.5 Correlation Heatmap Visualization

The correlation heatmap provides a visual summary of the correlation matrix.



In the heatmap:

- Red shades represent strong positive correlation
- Blue shades represent negative correlation
- Annotated values provide exact correlation coefficients

The heatmap clearly shows strong relationships between FWI and BUI, ISI, FFMC, DMC, and DC. This visualization confirms that FWI depends heavily on other fire indices.

This insight is important for feature selection and helps avoid redundancy during model training.

2.6 Feature Distribution Analysis

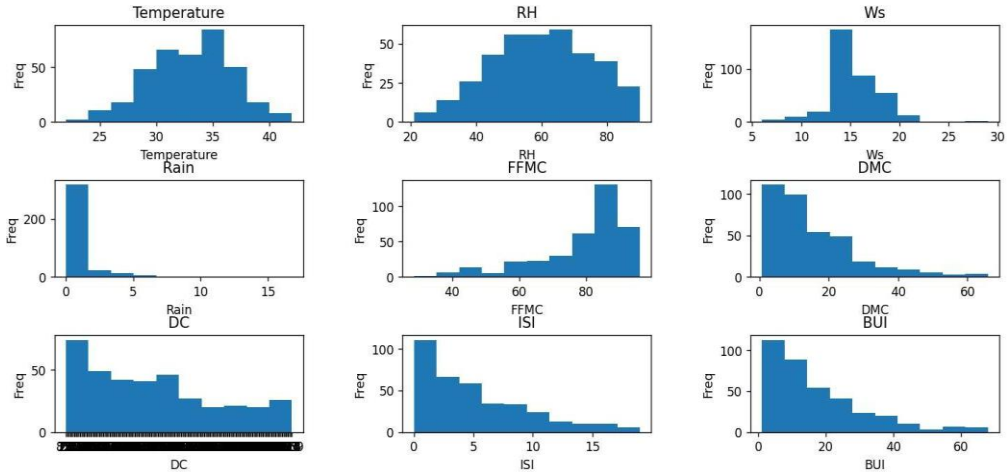
- To understand how individual features are distributed, histograms are plotted for major numerical attributes. Distribution analysis reveals skewness, spread, and concentration of values.
- Understanding distributions helps in detecting outliers, deciding normalization techniques, and interpreting model behavior.

2.7 Histogram Plotting Code

```
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_csv("dataset.csv")
cols = ["Temperature","RH","Ws","Rain","FFMC","DMC","DC","ISI","BUI"]
plt.figure(figsize=(14,12))
for i, col in enumerate(cols, 1):
    plt.subplot(4,3,i)
    plt.hist(df[col])
    plt.title(col)
    plt.xlabel(col)
    plt.ylabel("Freq")
    plt.subplots_adjust(wspace=0.4, hspace=0.6)

plt.show()
```

2.8 Interpretation of Histograms



- Temperature shows a moderately narrow distribution, indicating stable climate conditions across observations.
- Relative Humidity values are spread across a wide range, suggesting varied moisture conditions.
- Wind Speed values show moderate concentration with fewer extreme values.
- Rain distribution is highly skewed, with many zero or low rainfall values. This indicates dry conditions dominate the dataset.
- FFMC, DMC, DC, ISI, and BUI show skewed distributions with higher concentration toward lower values and fewer high-intensity observations. This pattern reflects real-world fire behavior where extreme conditions occur less frequently.
- These distributions indicate that normalization or scaling may improve model performance.

2.9 Importance of Dataset Statistics

Statistical analysis provides critical insights before model development. Correlation analysis identifies influential features and dependencies. Distribution analysis reveals variability and potential preprocessing needs.

These statistics help:

- Reduce redundant features
- Improve model stability
- Enhance interpretability
- Support accurate predictions

This section fulfills dataset statistics requirements and prepares the foundation for model building and evaluation.

SECTION 3

RIDGE REGRESSION MODEL

3.1 Introduction to Ridge Regression

Ridge Regression is a regularized linear regression technique used to handle multicollinearity and prevent overfitting. In datasets where input features are highly correlated, standard linear regression produces unstable coefficients. Ridge Regression solves this issue by adding an L2 penalty term to the loss function.

In this project, Ridge Regression is used to predict the Fire Weather Index (FWI), which is a continuous numerical variable. Since FWI depends on multiple interrelated fire indices such as FFMC, DMC, DC, ISI, and BUI, Ridge Regression is well suited for this task.

3.2 Objective of the Model

The primary objective of this section is to build a regression model that accurately predicts FWI based on selected weather and fire-related features.

Specific goals include:

- Reducing overfitting using regularization
- Identifying the optimal regularization parameter alpha
- Evaluating model performance using standard regression metrics
- Saving the trained model and scaler for future use

3.3 Data Preparation for Ridge Regression

Before training the Ridge Regression model, the dataset undergoes additional preprocessing to ensure numerical consistency.

Column names are cleaned by stripping extra spaces to avoid referencing errors. Selected numerical columns are explicitly converted to floating-point values. Any non-convertible values are replaced with NaN and later removed.

Rows containing missing values after conversion are dropped to maintain numerical integrity during training.

This preparation ensures that the regression model receives clean and well-structured input data.

3.4 Feature Selection and Target Variable

The following features are selected as input variables:

- FFMC
- DMC
- DC
- ISI
- Temperature
- Ws
- BUI

These features are chosen because they directly influence fire intensity and fire spread behavior.

The target variable is:

- FWI

FWI represents the overall fire intensity and is predicted as a continuous value.

3.5 Ridge Regression Implementation Code

The following Python code is used to implement Ridge Regression:

```
import pandas as pd
import numpy as np
import pickle
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import matplotlib.pyplot as plt

# Load dataset
df = pd.read_csv(r"D:\wfi\FWI-Prediction\dataset.csv")

# Clean column names
df.columns = df.columns.str.strip()

# Convert numeric columns
numeric_cols = ['FFMC','DMC','DC','ISI','Temperature','Ws','BUI','FWI']
for col in numeric_cols:
    df[col] = pd.to_numeric(df[col].astype(str).str.strip(), errors='coerce')

# Drop rows with NaN
df = df.dropna()

# Features and target
X = df[['FFMC','DMC','DC','ISI','Temperature','Ws','BUI']].values
```

```
y = df['FWI'].values

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Feature scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Save scaler
with open("scaler.pkl","wb") as f:
    pickle.dump(scaler, f)
```

3.6 Importance of Feature Scaling

Ridge Regression is sensitive to feature magnitude. Features with larger scales can dominate the penalty term and distort learning.

StandardScaler standardizes features to zero mean and unit variance. This ensures that all input variables contribute equally to the model.

Scaling improves numerical stability and leads to better convergence during optimization.

3.7 Hyperparameter Tuning Using Alpha

The alpha parameter controls the strength of regularization.

Small alpha values allow the model to fit closely to the data.
Large alpha values increase regularization and simplify the model.

Multiple alpha values are tested to identify the optimal balance between bias and variance.

Alpha values tested:

- 0.01
- 0.1
- 1
- 10
- 100

For each alpha, the model is trained and evaluated on both training and testing data.

3.8 Model Evaluation Metrics

The following metrics are used to evaluate model performance:

Mean Squared Error (MSE)
Measures average squared prediction error. Lower values indicate better performance.

Mean Absolute Error (MAE)
Measures average absolute difference between actual and predicted values.

Root Mean Squared Error (RMSE)
Square root of MSE. Interpretable in the same units as FWI.

R-squared Score (R2)
Measures how well model explains variance in the target variable. Higher values indicate better fit.

3.9 Alpha Tuning and Best Model Selection Code

```
alpha_values = [0.01, 0.1, 1, 10, 100]

mse_train_list, mse_test_list = [], []
mae_list, rmse_list = [], []

best_r2 = -np.inf
best_alpha = None

for alpha in alpha_values:
    model = Ridge(alpha=alpha)
    model.fit(X_train, y_train)

    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)

    mse_train = mean_squared_error(y_train, y_train_pred)
    mse_test = mean_squared_error(y_test, y_test_pred)
    mae = mean_absolute_error(y_test, y_test_pred)
    rmse = np.sqrt(mse_test)
    r2 = r2_score(y_test, y_test_pred)

    mse_train_list.append(mse_train)
    mse_test_list.append(mse_test)
    mae_list.append(mae)
    rmse_list.append(rmse)

    if r2 > best_r2:
        best_r2 = r2
        best_alpha = alpha
        best_model = model

# Save best model
with open("ridge.pkl","wb") as f:
    pickle.dump(best_model, f)
```

3.10 Performance Visualization

To visualize the effect of alpha on performance, multiple plots are generated.

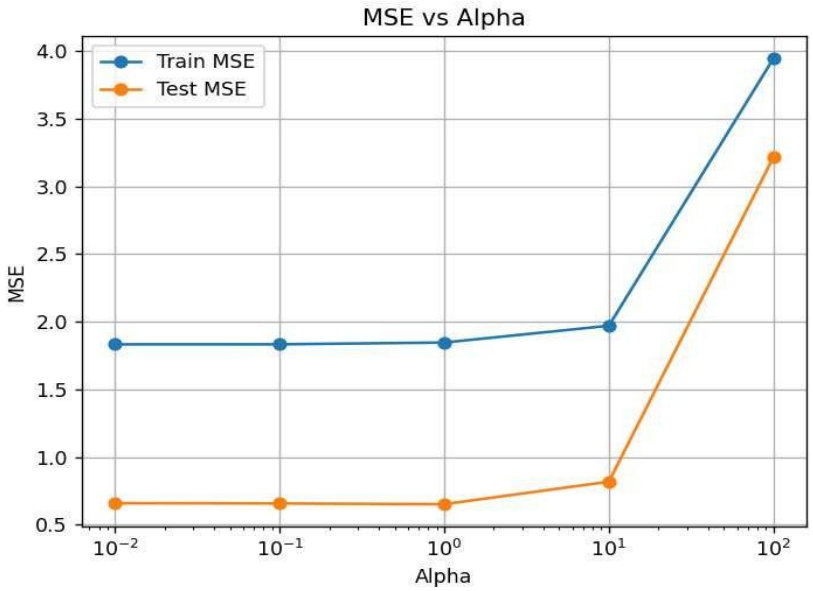


Figure 3.1: MSE vs Alpha

Shows taining and testing error variation across alpha values.

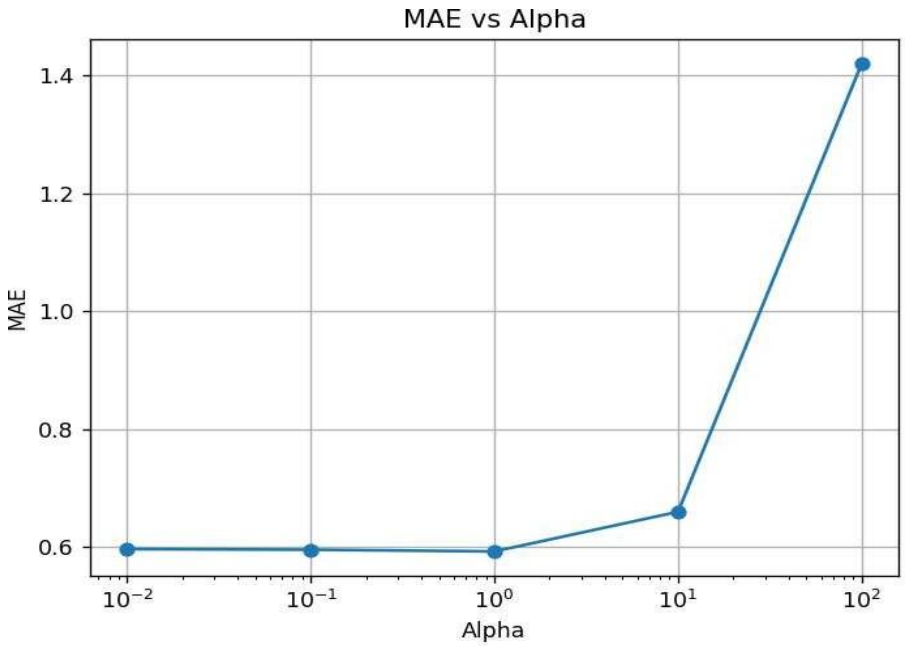


Figure 3.2: MAE vs Alpha

Shows how absolute error changes with regularization strength.

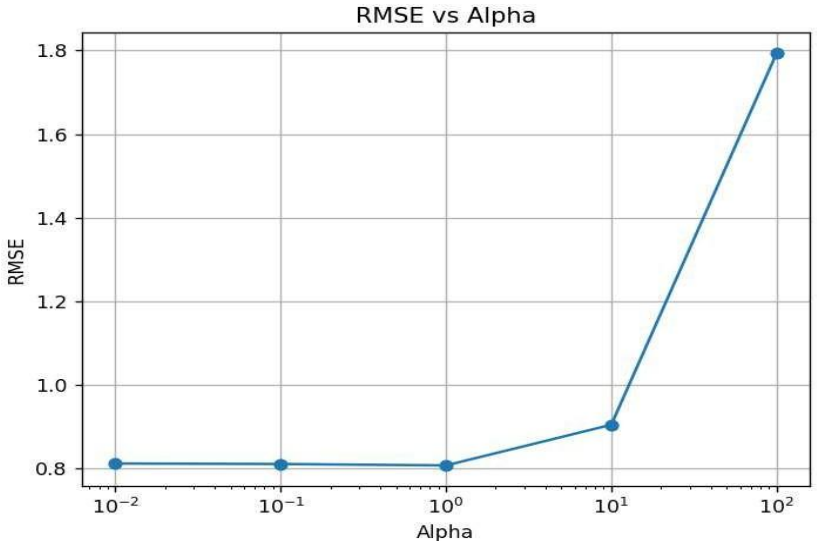


Figure 3.3: RMSE vs Alpha

Shows prediction error magnitude across alpha values.

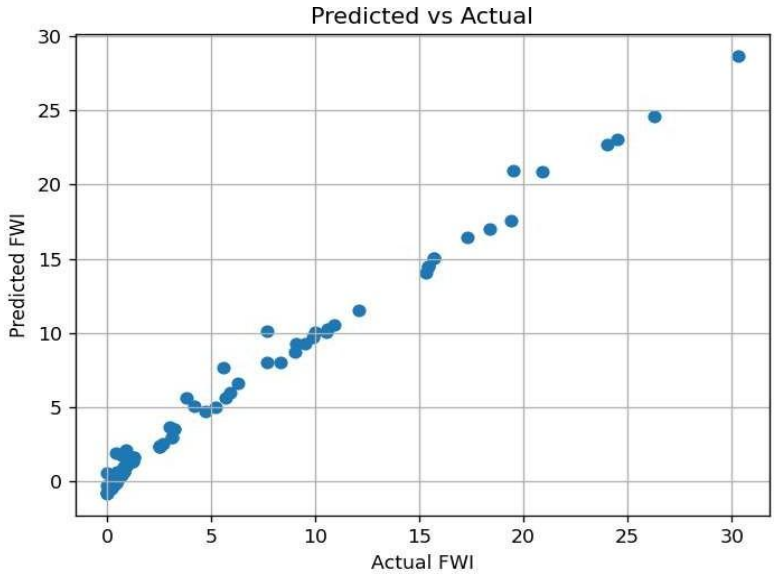
```
plt.figure()
plt.plot(alpha_values, mse_train_list, marker='o', label="Train MSE")
plt.plot(alpha_values, mse_test_list, marker='o', label="Test MSE")
plt.xscale('log')
plt.xlabel("Alpha")
plt.ylabel("MSE")
plt.title("MSE vs Alpha")
plt.legend()
plt.grid(True)
plt.show()

plt.figure()
plt.plot(alpha_values, mae_list, marker='o')
plt.xscale('log')
plt.xlabel("Alpha")
plt.ylabel("MAE")
plt.title("MAE vs Alpha")
plt.grid(True)
plt.show()

plt.figure()
plt.plot(alpha_values, rmse_list, marker='o')
plt.xscale('log')
plt.xlabel("Alpha")
plt.ylabel("RMSE")
plt.title("RMSE vs Alpha")
plt.grid(True)
plt.show()
```

3.11 Predicted vs Actual Analysis

A scatter plot is used to compare predicted FWI values with actual FWI values.



Points close to the diagonal line indicate accurate predictions. This plot helps visually assess model accuracy and bias.

```
y_pred_best = best_model.predict(X_test)
plt.figure()
plt.scatter(y_test, y_pred_best)
plt.xlabel("Actual FWI")
plt.ylabel("Predicted FWI")
plt.title("Predicted vs Actual")
plt.grid(True)
plt.show()
```

3.12 Summary of Ridge Regression Model

Ridge Regression successfully models the relationship between fire weather indices and FWI. Regularization reduces overfitting and improves generalization. Alpha tuning identifies the optimal model configuration.

The trained model and scaler are saved for reuse, making the system suitable for deployment and further experimentation.

SECTION 4

FLASK WEB APPLICATION AND MODEL DEPLOYMENT

4.1 Objective of the Flask Application

The objective of the Flask web application is to deploy the trained Ridge Regression model and provide an interactive interface for Fire Weather Index prediction. The application allows users to input fire-related parameters through a web form and obtain real-time FWI predictions along with a clearly defined fire danger level.

This section demonstrates the practical implementation of the machine learning model and completes the end-to-end workflow from data processing to deployment.

4.2 System Architecture

The system consists of two main components.

- The backend is developed using the Flask framework. It loads the trained Ridge Regression model and the StandardScaler saved during training. It handles user input, performs preprocessing, generates predictions, and sends results to the frontend.
- The frontend is developed using HTML and CSS. It provides a clean input form for feature entry and a result page that visually presents the predicted FWI and corresponding fire danger level.
- Static resources such as background images and fire danger icons are stored in the static directory.

4.3 Model and Scaler Integration

The trained Ridge Regression model and the feature scaler are stored as serialized pickle files. These files are loaded when the Flask application starts.

Using the same scaler during prediction ensures consistency between training data and user input. This step is essential to maintain prediction accuracy and reliability.

4.4 Flask Backend Implementation

The Flask backend manages routing, input handling, and prediction logic.

Key functionalities:

- Loads model and scaler
- Accepts user input through POST request
- Scales input features
- Predicts Fire Weather Index

- Classifies fire danger level
- Renders result page

Backend code used:

```
from flask import Flask, render_template, request
import numpy as np
import pickle

app = Flask(__name__)

with open("ridge.pkl", "rb") as f:
    model = pickle.load(f)

with open("scaler.pkl", "rb") as f:
    scaler = pickle.load(f)

@app.route("/")
def index():
    return render_template("index.html")

@app.route("/predict", methods=["POST"])
def predict():
    FFMC = float(request.form["FFMC"])
    DMC = float(request.form["DMC"])
    DC = float(request.form["DC"])
    ISI = float(request.form["ISI"])
    Temperature = float(request.form["Temperature"])
    Ws = float(request.form["Ws"])
    BUI = float(request.form["BUI"])

    data = np.array([[FFMC, DMC, DC, ISI, Temperature, Ws, BUI]])
    scaled = scaler.transform(data)

    prediction = round(model.predict(scaled)[0], 3)

    if prediction < 5:
        level = "LOW Fire Danger"
        image_file = "low.png"
    elif prediction < 12:
        level = "MODERATE Fire Danger"
        image_file = "moderate.png"
    elif prediction < 21:
        level = "HIGH Fire Danger"
        image_file = "high.png"
    elif prediction < 38:
        level = "VERY HIGH Fire Danger"
        image_file = "veryhigh.png"
    else:
        level = "EXTREME Fire Danger"
        image_file = "extreme.png"
```

```
    return render_template(
        "result.html",
        result=prediction,
        level=level,
        image_file=image_file
    )

if __name__ == "__main__":
    app.run(debug=True)
```

4.5 User Input Interface

The input interface is designed to collect all features required by the model.

Input parameters:

- FFMC
- DMC
- DC
- ISI
- Temperature
- Wind Speed
- BUI

The input page uses a visually appealing background and form layout to improve user experience. All fields are mandatory to avoid incomplete input.

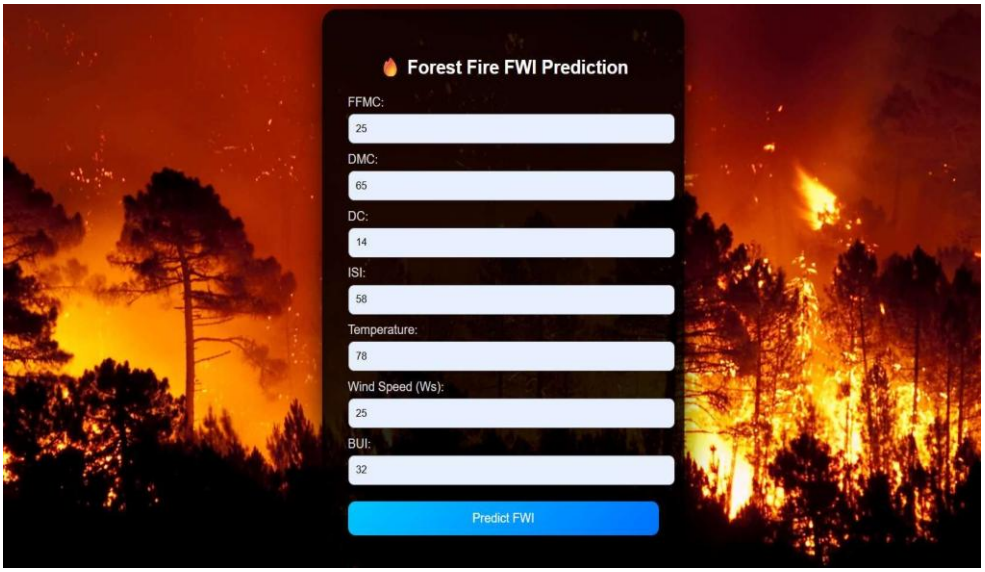


Figure : Flask Application Input Page

4.6 Input Page Implementation

The input page is implemented using HTML and CSS. The form submits user input to the /predict route using the POST method.

This page ensures correct data collection and smooth interaction between user and backend.

4.7 Prediction and Fire Danger Classification

Once input values are submitted, the backend scales the data and generates the predicted Fire Weather Index.

The predicted FWI is mapped into five fire danger categories:

- Low
- Moderate
- High
- Very High
- Extreme

This classification converts numerical output into an easily interpretable form for users.

4.8 Result Display Interface

The result page displays:

- Predicted FWI value
- Fire danger level
- Visual icon representing danger severity

Animations and color gradients are used to improve clarity and engagement.

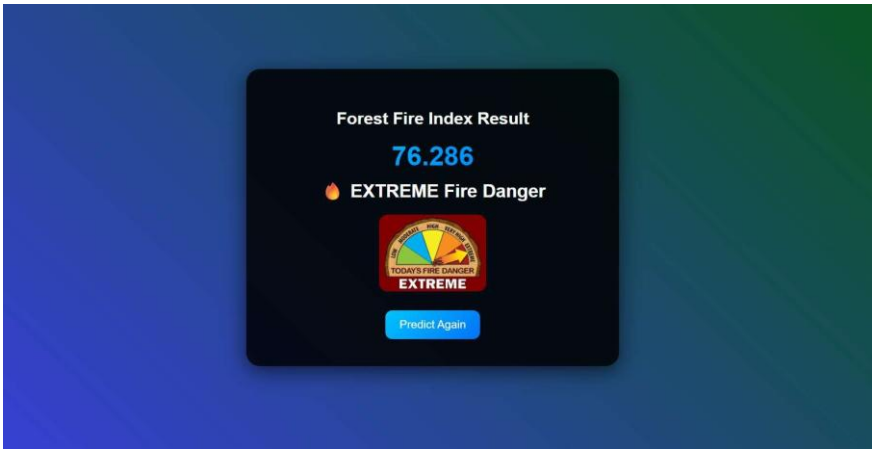


Figure : Flask Application Result Page

4.9 Demo Video Description

A demo video is recorded to demonstrate the complete working of the Flask application.

The video includes:

- Starting the Flask server
- Opening the application in a browser
- Entering sample input values
- Generating FWI prediction
- Displaying fire danger level

The demo video is uploaded to Google Drive for easy access.

Demo Video Link (Google Drive):

https://drive.google.com/file/d/1twhCZei8zdSlAkXbKVHGpGkfmneotoG/view?usp=sharing

---

4.10 Significance of Deployment

The Flask application validates the practical usability of the machine learning model. It enables real-time prediction, improves accessibility, and presents results in an understandable format.

This section completes the project lifecycle by integrating data preprocessing, model training, and real-world deployment into a single functional system.