ITS66604_0369922_INDVASGNMT

by SUSHANT TANDUKAR .

Submission date: 17-Jul-2025 07:30PM (UTC+0800)
Submission ID: 2716319437
File name: 241021_SUSHANT_TANDUKAR_._ITS66604_0369922_INDVASGNMT_760881_1251061704.pdf (1.29M)
Word count: 3882
Character count: 24167

TAYLOR'S UNIVERSITY
Wisdom · Integrity · Excellence

TAYLOR'S PROGRAMMES

MAY 2025 SEMESTER

ITS66604 - Machine Learning and Parallel Computing

Assignment No./Title	**Assignment (Individual Assignment)** **20% Weightage**
Course Tutor/Lecturer	Mr. Anmol Adhikari, Mr. Bidhan Chandra Bhattarai
Submission Date	17th July 2025
Student Name, ID and Signature 1. Sushant Tandukar– 0369922	

Declaration *(need to be signed by students. Otherwise, the assessment will not be evaluated)*

Certify that this assignment is entirely my own work, except where I have given fully documented references to the work of others, and that the material contained in this assignment has not previously been submitted for assessment in any other formal course of study.

Marks/Grade:	**Evaluated by:**
Evaluator's Comments:	

* Please include this cover page for your project submission

Table of Contents

1. Problem Context & Dataset Justification

Wildfires are among the most destructive natural hazards, especially in Mediterranean countries such as Turkey. Predicting and managing wildfires effectively calls for a thorough data analysis to identify underlying relationships and utilize this in decision making. This report intends to analyze the dataset "2000–2021 Turkey Forest Fires" coming from Kaggle (Dincer, 2021) assess its relation to wildfire risk modelling, and rationalize this dataset to be used for machine learning (ML) and parallel computing in.

1.1 Problem Context

Turkey's great vulnerability to forest fires, especially during the dry summer months, means that larger events, such as the wildfires in southern provinces in 2021, cause severe ecological and economic losses. Quantifying and predicting wildfire behavior is critical for:

- Early detection and evacuation planning
- Efficient and effective use of firefighting resources
- Policies for sustainable forest management
- Public education and preparedness

Wildfire data are paramount for computational methods that accomplish these aims.

1.2 Dataset Justification

The dataset selected, 2000-2021SINGLE TURKEY FIRE_M-C61_214067.csv, contains fire incident records from Turkey from 2000-2021. The dataset contains attribute data of:

- Latitude, Longitude – Spatial location
- Brightness – Thermal energy from satellite sensors
- Scan and Track – Satellite sensor resolution in x and y directions
- Confidence – Confidence on detection between 0-100%
- FRP – Fire intensity variable
- Bright_t31 – Brightness temperature at 31μm
- Acquisition Date & Time – When the fire was detected

Here Bright_t31 represents Brightness temperature at 31μm (31 micrometers) since it's a section of the infrared range that is sensitive to surface heat.(Dincer, 2021)

1.2.1 Importance of Dataset

This dataset is very valuable for a number of reasons:

- **Disaster Management and Preparedness:** Decision-makers can act upon historical fire events to improve disaster management, and allocate resources in the most effective way when combating wildfires. (nfpa.org, n.d.)
- **Climate Change Impact Assessment:** This dataset allows analysis of climate linkages with fire episodes over the long history of the dataset, and further research in Turkey about the impacts of climate change on fire regimes.
- **Scientific Research:** This dataset could be used to create and validate prediction models for wildfires, evaluate efficiency of prevention strategies, and explore the socio-economic processes related to wildfires.

1.2.2 Importance in Machine Learning and Parallel Computing

This data is very well-suited for Parallel computing and Machine Learning (ML) based on the data:

Suitable for Machine Learning

- **Predictive Modeling:** The dataset contains features (such as latitude, longitude, brightness, date/timestamp of acquisition and more) that can predict different aspects of fire incidents such as predicting the fire ignition, predicting where fire will occur, predicting fire severity, and impending fire fighter action. (Remote Sensing of Environment, n.d.)
- **Classification problem and regression problem:** Since the dataset includes target variables that are available to use for classification (such as prediction of fire type, day or night prediction of fire) and regression tasks (such as prediction of darkness, FRP value), the dataset can be used for both machine learning tasks.
- **Time Series Analysis:** The acq_time and acq_date fields enable time series analysis, which would enable more insight into seasonality, long-term trends, and deviation with regard to fire. This is necessary in order to be able to predict future fires.

Suitable for Parallel computing

- **Huge Amount of Data:** Even though the number of records is not provided, an n-year record of fire incidents for an entire country is likely to be very large and processing such

a volume of data can be computationally intensive. Parallel computing can divide the task among multiple processors or nodes significantly reducing processing time. (Apache spark: A unified engine for big data processing)

- **Independent Data Points:** Every fire incident record in the dataset can (for the most part) be treated as an independent data point for the majority of ML algorithms. This renders the dataset highly amenable to parallel processing because computations for a particular record do not typically rely on the results of computations for other records. (MapReduce: simplified data processing on large clusters)
- **Iterative Model Training:** The majority of ML algorithms, particularly those utilized in deep learning or ensemble methods (e.g., Random Forests), involve iterative training processes or training multiple sub-models. These sub-models or iterations can very often be run in parallel which leads to faster creation of model. (Guolin Ke)

The challenges in this data set are:

- Data Imbalance: Most fires are small with low FRP, but there are some extreme cases that dominate the data
- Outliers: The extreme values in the data need proper treatment
- Noise: Because of variation in scan and track values, it may contain noise because of sensor limitations in satellite
- Computation Power: If this data is combined with weather, temperature the dataset will need more computation power.

Data exploration and Preprocessing

- For handling missing values in datasets, we dropped the rows with missing values

```
# Handle missing values if any
df = df.dropna()
```

Figure 1: Handling missing values

- Removed outliers by using IQR

IQR = Q3-Q1

Q3 = median of 3rd quartile

Q1 = median of 1st quartile

To find the outliers,

Lower bound = Q1 - 1.5 * IQR

Upper bound = Q3 + 1.5 * IQR

```
# Remove outliers using IQR
# Calculate Q1 and Q3 for numerical columns
numerical_cols = df.select_dtypes(include=np.number).columns.tolist()

Q1 = df[numerical_cols].quantile(0.25)
Q3 = df[numerical_cols].quantile(0.75)
IQR = Q3 - Q1

# Define bounds for outliers
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Filter out outliers
df_filtered = df[~((df[numerical_cols] < lower_bound) | (df[numerical_cols] > upper_bound)).any(axis=1)]

print(f"Original data shape: {df.shape}")
print(f"Filtered data shape: {df_filtered.shape}")

df = df_filtered.copy()
```

```
Original data shape: (205918, 15)
Filtered data shape: (133412, 15)
```

Figure 2: removing outliers

If anything is below lower bound or above upper bound, it is an outlier and is removed

- Generated a correlation heatmap to find relationships between columns which may be helpful in finding FRP

```
# Compute correlation matrix
corr = df[['latitude', 'longitude', 'brightness', 'scan', 'track', 'bright_t31', 'confidence', 'frp']].corr()

# Plot heatmap
plt.figure(figsize=(8,6))
sns.heatmap(corr, annot=True, cmap='coolwarm', fmt=".2f", square=True, linewidths=0.5)
plt.title("Feature Correlation Matrix")
plt.show()
```

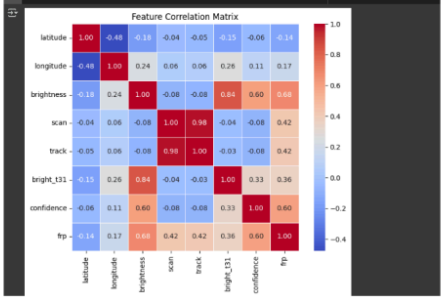


Figure 3: Generating Correlation matrix

- Selecting columns of features to predict FRP (brightness, scan, track, and confidence)

```
# Select features and target

# Features: Use numerical columns only
features = ['brightness', 'scan', 'track','confidence']
target = 'frp'

X = df[features]
y = df[target]
```

Figure 4: Features selection

2. System Design for Scalable Machine Learning Solution

2.1 Model Selection

To analyze and predicting wild fire patterns from the dataset, I choose most commonly used machine learning methods, **Linear Regression and Random Forest Regressor**. In general, these methods offer a balance of model interpretability and predictive power. Therefore, they are considered a sensible choice for working with structured tabular data such as wildfire records.

Evaluation Matrices:

Mean Absolute Error (MAE): MAE is the average of absolute difference of actual and predicted value.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Figure 5: Mean Absolute error formula

Where,

n = number of observations

y_i = true value

$\hat{y}_i$ = predicted value

Root Mean Squared Error (RMSE): RMSE measures the average magnitude of errors.

$$\mathrm{RMSE} = \sqrt{\frac{\sum_{i=1}^{N} \|y(i) - \hat{y}(i)\|^2}{N}}$$

Figure 6: RMSE formula

Where,

n = number of observations

y_i = true value

$\hat{y}_i$ = predicted value (What is Root Mean Square Error?, n.d.)

R^2: R^2 shows the percentage of variance in the actual value that is explained by the model. It ranges from 0-1.

$$R^2 = 1 - \frac{SS_{RES}}{SS_{TOT}} = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \overline{y})^2}$$

Figure 7:R^2 formula

Where,

n = number of observations

y_i = true value

$\hat{y}_i$= predicted value (Kumar, 2023)

- **Linear Regression**

Linear regression is a supervised learning algorithm for modeling the relationship between a dependent variable (target) and one or more independent variables (features), and is based on the assumption that it is a linear relationship between the inputs and output. (Linear regression in machine learning, 2025) Linear Regression will form a baseline for the FRP prediction. The drawback for linear regression is that it is less robust for outliers and complex patterns. Since we are using multiple linear regression:

Formula multiple linear regression: $h(x_1,x_2,\ldots,x_k)=\beta_0+\beta_1x_1+\beta_2x_2+\ldots+\beta_kx_k$

Where:

- $x_1, x_2, \ldots, x_k$ $x_1, x_2, \ldots, x_k$ are the independent variables.
- β_0 β_0 is the intercept.
- $\beta_1, \beta_2, \ldots, \beta_k$ $\beta_1, \beta_2, \ldots, \beta_k$ are the coefficients, representing the influence of each respective independent variable on the predicted output. (Linear regression in machine learning, 2025)

For our case: FRP= β0 + β1 × brightness + β2 × scan + β3× track + β4 ×confidence

Where, e = error term

Python Code:

```
[ ]  # Select features and target

     # Features: use numerical columns only
     features = ['brightness', 'scan', 'track','confidence']
     target = 'frp'

     X = df[features]
     y = df[target]
```

Figure 8: Features selection

```
     # Split data into train and test sets
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Figure 9: spliting data

```
[ ]  # Linear Regression Model
     lr = LinearRegression()
     lr.fit(X_train, y_train)

     # Predict
     y_pred_lr = lr.predict(X_test)

     # Evaluate Linear Regression
     print("Linear Regression Performance:")
     mae_lr_sequential = mean_absolute_error(y_test, y_pred_lr)
     mse_lr_sequential = mean_squared_error(y_test, y_pred_lr)
     rmse_lr_sequential = np.sqrt(mean_squared_error(y_test, y_pred_lr))
     r2_lr_sequential = r2_score(y_test, y_pred_lr)
     print("MAE:", mae_lr_sequential)
     print("MSE:", mse_lr_sequential)
     print("RMSE:", rmse_lr_sequential)
     print("R2 Score:", r2_lr_sequential)

     Linear Regression Performance:
     MAE: 4.727006291029031
     MSE: 38.125269880787506
     RMSE: 6.174566371882928S
     R2 Score: 0.7549565415424608
```

Figure 10: Linear model code

```python
import statsmodels.api as sm

# Linear Regression Model Summary
X_train_sm = sm.add_constant(X_train) # Add a constant for the intercept
lr_sm = sm.OLS(y_train, X_train_sm).fit()
print("Linear Regression Model Summary:")
print(lr_sm.summary())
```

```
Linear Regression Model Summary:
                            OLS Regression Results
==============================================================================
Dep. Variable:                    frp   R-squared:                       0.760
Model:                            OLS   Adj. R-squared:                  0.760
Method:                 Least Squares   F-statistic:                 4.241e+04
Date:                Wed, 16 Jul 2025   Prob (F-statistic):               0.00
Time:                        12:03:05   Log-Likelihood:            -1.7638e+05
No. Observations:               53633   AIC:                         3.528e+05
Df Residuals:                   53628   BIC:                         3.528e+05
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       -228.2931      1.112   -205.230      0.000    -230.473    -226.113
brightness     0.6227      0.003    195.364      0.000       0.617       0.629
scan          13.6186      0.332     40.988      0.000      12.967      14.270
track          7.5981      0.889      8.550      0.000       5.856       9.340
confidence     0.2753      0.002    116.467      0.000       0.271       0.280
==============================================================================
Omnibus:                     7244.648   Durbin-Watson:                   2.009
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            11886.655
Skew:                           0.929   Prob(JB):                         0.00
Kurtosis:                       4.367   Cond. No.                     1.51e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.51e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Figure 11: Ols regression model summary for Linear regression

```python
# Plot Linear Regression Results
plt.figure(figsize=(8,6))
sns.scatterplot(x=y_test, y=y_pred_lr, alpha=0.5, color="blue")
# Plot perfect prediction line
min_val = min(y_test.min(), y_pred_lr.min())
max_val = max(y_test.max(), y_pred_lr.max())
plt.plot([min_val, max_val], [min_val, max_val], '--', color='red', label='Perfect Prediction Line')
plt.xlabel("Actual FRP")
plt.ylabel("Predicted FRP (Linear Regression)")
plt.title("Linear Regression: Actual vs Predicted")
plt.show()
```

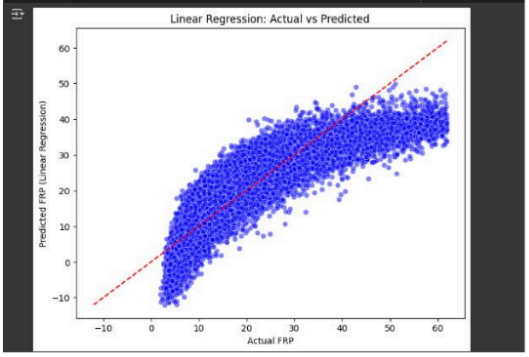


Figure 12: Linear model result

Interpretation:

- MAE = 4.72 : Essentially, on average the model's predictions are off by 4.72 units.
- RMSE: An RMSE of 6.17 indicates the model's predictions were 6.17 units away from the actual value on average. A lower RMSE indicates the model is more accurate.
- R2: This suggests that the model explains roughly 75.4% of the variance in area burned or FRP.

- **Random Forest**

Random Forest is a supervised learning algorithm for building many decision trees and merging their predictions for enhanced accuracy and reliability. Random Forest can be capable of identifying hard, non-linear relationships in the data. The drawback for random forest regression, it is computationally expensive and slow to train especially with numerous trees and high-dimensional data. (Random Forest Regression in Python, 2025)

Python code:

```
# Random Forest Model
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

# Predict
y_pred_rf = rf.predict(X_test)

# Evaluate Random Forest
print("Random Forest Performance:")
mae_rf_sequential = mean_absolute_error(y_test, y_pred_rf)
mse_rf_sequential = mean_squared_error(y_test, y_pred_rf)
rmse_rf_sequential = np.sqrt(mean_squared_error(y_test, y_pred_rf))
r2_rf_sequential = r2_score(y_test, y_pred_rf)
print("MAE:", mae_rf_sequential)
print("MSE:", mse_rf_sequential)
print("RMSE:", rmse_rf_sequential)
print("R2 Score:", r2_rf_sequential)
```

```
Random Forest Performance:
MAE: 2.822545196205912
MSE: 16.73594800292295
RMSE: 4.090959301059221
R2 Score: 0.8924326413419457
```

Figure 13: Random forest code

Interpretation:

- MAE: This indicates that, on average, the model predictions are off by 2.82 units.
- RMSE: This indicates that, on average, the prediction errors are 4.09 units away from the true values, and since it's squared, bigger errors are more severely penalized.

- R2: This indicates that 89.2% of the variance in the target variable is accounted for by the model.

```
[ ] importances = rf.feature_importances_
    features = X.columns

    # Show feature importances
    for feature, importance in zip(features, importances):
        print(f"{feature}: {importance:.4f}")

brightness: 0.1537
scan: 0.1644
track: 0.1450
confidence: 0.5369
```

Figure 14: Importance of features

Interpretation:

- The confidence feature has the greatest effect (importance = 0.5369), contributing more than 50% to the predictions.
- Scan, brightness, and track have moderate effect with track slightly less in importance.

```
# Plot Random Forest Results
plt.figure(figsize=(8,6))
sns.scatterplot(x=y_test, y=y_pred_rf, alpha=0.5, color="green")
plt.plot([min_val, max_val], [min_val, max_val], '--', color='red', label='Perfect Prediction Line')
plt.xlabel("Actual FRP")
plt.ylabel("Predicted FRP (Random Forest)")
plt.title("Random Forest: Actual vs Predicted")
plt.show()
```

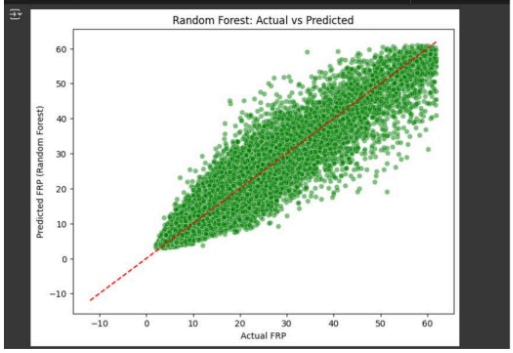


Figure 15: Random forest result

2.2 Parallel Computing

To handle potentially overwhelming levels of fire data and accelerate model training and prediction, parallel computation libraries are required. I will utilize Apache Spark through PySpark for distributed computation and Python's multiprocessing library for local-level parallelism.

Spark / PySpark

Apache Spark is an open-source, distributed high-performance processing engine used in big data workload. It supports in-memory computing capabilities, and hence it is far more rapid compared to traditional disk-based processing engines like Hadoop MapReduce (Apache Spark, n.d.). PySpark is the Python API of Spark, allowing data scientists and engineers to leverage Spark using standard Python syntax.

Advantages

- **Distributed Data Processing:** Spark can spread large datasets over a cluster of machines to allow parallel execution of data cleaning, transformation, and feature engineering tasks. That's what it takes to handle decades of fire data. (Apache Spark, n.d.)
- **Scalability:** With an increase in data set, Spark can scale horizontally by the simple method of adding nodes to the cluster, hence keeping processing time under control. (Apache Spark, n.d.)

Python Multi-processing

While Spark addresses distributed processing on a cluster, Python's multiprocessing library aids parallelism on a single machine by leveraging multiple CPU cores. This is particularly useful for tasks that can be divided into independent sub-tasks and run in parallel on a multi-core processor. This is particularly useful for tasks that can be divided into independent sub-tasks and run concurrently on a multi-core processor. (Multiprocessing - process-based parallelism, n.d.)

Advantages:

- **Data Pre-processing:** Certain pre-processing steps, e.g., parsing specific columns, applying custom functions row-wise, or generating complex features, can be parallelized with multiprocessing. Pool to distribute the work to available CPU cores.

- **Simplicity for Local Jobs:** For those jobs that can fit in the memory of one machine, multiprocessing is generally simpler to set up and arrange.

Code of Parallel Computing:

- Calling Library, Spark Session creation and Dataframe creation

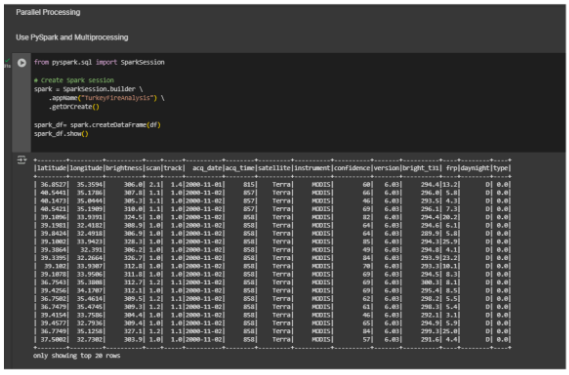


Figure 16: Calling Spark Session Library, creating spark session and dataframe creation

- Using sum() function to calculate total brightness and showing final result

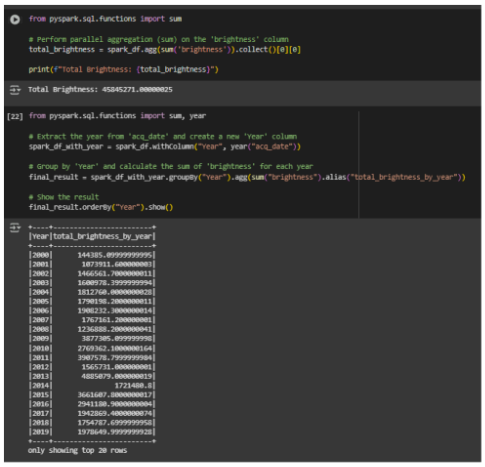


Figure 17: Calculation total brightness and showing final multi-processing result

2.3 System overview

- **Data Loading and Initial Inspection:** The system starts with loading wildfire data from a CSV file, preliminary checks were done on data such as printing out few rows and look at data types and missing values.
- **Preprocessing of data:**
  - The system checks for missing values and drops rows that have any missing values. Outliers are identified and removed using Interquartile Range (IQR) approach which was done to increase the accuracy of the models.
  - Selecting features: Some of the numerical columns were selected to act as features for training model.
- **Data Splitting:** Dataset was divided into train and test sets. This step is essential for testing performance of the model with unseen data.
- **Training Model:**
  - Linear Regression: Linear Regression model was created and trained with the training set.
  - Random Forest: Random Forest Regressor was instantiated and trained with the training set.

- **Model Evaluation:**
  - The performance of the models is verified against several regression metrics: Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R-squared (R2 Score).
- **Prediction:**
  - Predictions are made on the test data (X_test) by the trained models. Parallel processing using joblib is applied to the prediction process of Linear Regression and is natively supported by the n_jobs parameter in RandomForestRegressor.
- **Results Comparison and Visualization:**
  - The performance metrics and training times for sequential and parallel processing are compiled into a pandas DataFrame for easy comparison. Bar plots are drawn to represent the comparisons of training time and R2 Score between the models and processing types.

In summary, the system loads and prepares the wildfire data, selects features, splits the data, trains Linear Regression and Random Forest models (with and without parallel processing), makes predictions, and finally evaluates and visualizes the performance and training times of the models.

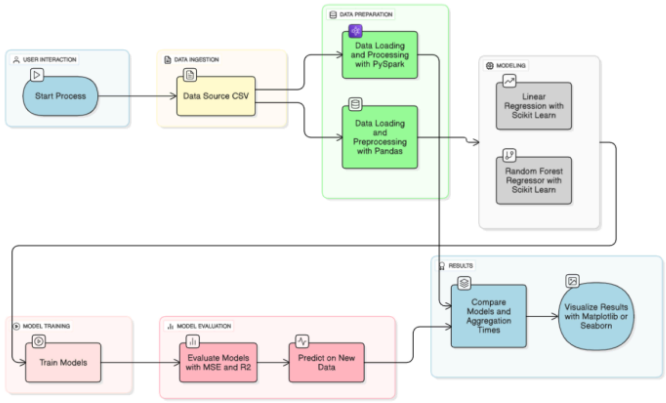


Figure 18: System Diagram (Eraser, n.d.)

3. Implementation Strategy

The current section describes the workflow used to train and evaluate machine learning models with sequences and parallel processing.

1. Splitting Data

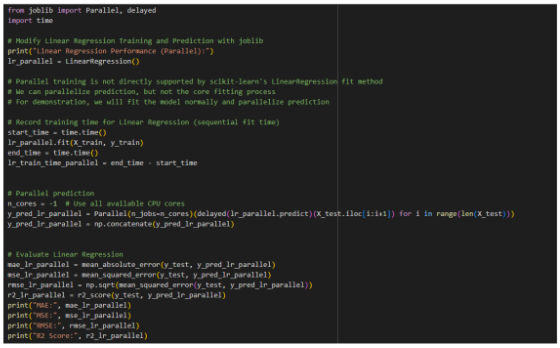
The dataset was split into training and testing datasets using train_test_split function. This is a fundamental step in machine learning which reduces the chances of overfitting and evaluate generalization performance of model while processing unseen data.

- Train: 80%
- Test: 20%

2. Module Training (Parallel)

Linear Regression

- Library: LinearRegression from Scikit-learn
- Training: Sequential only — n_jobs not supported for .fit()
- Workaround: While training is sequential, prediction can be simulated in parallel by chunking X_test and employing joblib.Parallel. Model fitting could be single-threaded, but we can apply n_jobs with joblib during prediction to parallelize over CPU cores.



```
from joblib import Parallel, delayed
import time

# Modify Linear Regression Training and Prediction with joblib
print("Linear Regression Performance (Parallel):")
lr_parallel = LinearRegression()

# Parallel training is not directly supported by scikit-learn's LinearRegression fit method
# We can parallelize prediction, but not the core fitting process
# For demonstration, we will fit the model normally and parallelize prediction

# Record training time for Linear Regression (sequential fit time)
start_time = time.time()
lr_parallel.fit(X_train, y_train)
end_time = time.time()
lr_train_time_parallel = end_time - start_time


# Parallel prediction
n_cores = -1  # Use all available CPU cores
y_pred_lr_parallel = Parallel(n_jobs=n_cores)(delayed(lr_parallel.predict)(X_test.iloc[i:i+1]) for i in range(len(X_test)))
y_pred_lr_parallel = np.concatenate(y_pred_lr_parallel)


# Evaluate Linear Regression
mae_lr_parallel = mean_absolute_error(y_test, y_pred_lr_parallel)
mse_lr_parallel = mean_squared_error(y_test, y_pred_lr_parallel)
rmse_lr_parallel = np.sqrt(mean_squared_error(y_test, y_pred_lr_parallel))
r2_lr_parallel = r2_score(y_test, y_pred_lr_parallel)
print("MAE:", mae_lr_parallel)
print("MSE:", mse_lr_parallel)
print("RMSE:", rmse_lr_parallel)
print("R2 Score:", r2_lr_parallel)
```

Random Forest Regressor

- Tool: RandomForestRegressor from Scikit-learn

- Parallel Training: Scikit-learn has native support for parallel training using n_jobs=-1 (all CPU cores). Parallelizes tree-building tasks and predictions internally.

```
print("\nRandom Forest Performance (Parallel):")
# Modify Random Forest Training and Prediction with joblib
# RandomForestRegressor supports n_jobs for parallel training
# Record training time for Random Forest (parallel)
start_time = time.time()
rf_parallel = RandomForestRegressor(n_estimators=100, random_state=42, n_jobs=n_cores)
rf_parallel.fit(X_train, y_train)
end_time = time.time()
rf_train_time_parallel = end_time - start_time


# Parallel prediction (RandomForestRegressor's predict also supports n_jobs)
y_pred_rf_parallel = rf_parallel.predict(X_test)


# Evaluate Random Forest
mae_rf_parallel = mean_absolute_error(y_test, y_pred_rf_parallel)
mse_rf_parallel = mean_squared_error(y_test, y_pred_rf_parallel)
rmse_rf_parallel = np.sqrt(mean_squared_error(y_test, y_pred_rf_parallel))
r2_rf_parallel = r2_score(y_test, y_pred_rf_parallel)
print("MAE:", mae_rf_parallel)
print("MSE:", mse_rf_parallel)
print("RMSE:", rmse_rf_parallel)
print("R2 Score:", r2_rf_parallel)
```

Module Training (Normal)

```
# Linear Regression Model
lr = LinearRegression()
lr.fit(X_train, y_train)

# Predict
y_pred_lr = lr.predict(X_test)

# Evaluate Linear Regression
print("Linear Regression Performance:")
mae_lr_sequential = mean_absolute_error(y_test, y_pred_lr)
mse_lr_sequential = mean_squared_error(y_test, y_pred_lr)
rmse_lr_sequential = np.sqrt(mean_squared_error(y_test, y_pred_lr))
r2_lr_sequential = r2_score(y_test, y_pred_lr)
print("MAE:", mae_lr_sequential)
print("MSE:", mse_lr_sequential)
print("RMSE:", rmse_lr_sequential)
print("R2 Score:", r2_lr_sequential)
```

```
Linear Regression Performance:
MAE: 4.727006291029031
MSE: 38.125269880787506
RMSE: 6.1745663718829285
R2 Score: 0.7549565415424608
```

```python
# Random Forest Model
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

# Predict
y_pred_rf = rf.predict(X_test)

# Evaluate Random Forest
print("Random Forest Performance:")
mae_rf_sequential = mean_absolute_error(y_test, y_pred_rf)
mse_rf_sequential = mean_squared_error(y_test, y_pred_rf)
rmse_rf_sequential = np.sqrt(mean_squared_error(y_test, y_pred_rf))
r2_rf_sequential = r2_score(y_test, y_pred_rf)
print("MAE:", mae_rf_sequential)
print("MSE:", mse_rf_sequential)
print("RMSE:", rmse_rf_sequential)
print("R2 Score:", r2_rf_sequential)
```

```
Random Forest Performance:
MAE: 2.822545196205912
MSE: 16.73594800292295
RMSE: 4.090959301059221
R2 Score: 0.8924326413419457
```

3. **Parallel Processing Considerations**
   - **Multiprocessing (CPU):** Prediction and RandomForestRegressor model training were parallelized through CPU-based multiprocessing using the aid of the n_jobs parameter of Scikit-learn. In LinearRegression, although training could not be parallelized, predictions were hand-parallelized using joblib, highlighting an alternative way of benefiting from multi-core systems.
   - **GPU Acceleration:** No parallelization on GPU was performed. Scikit-learn primarily uses CPU for parallel executions. For machine learning that is GPU-accelerated (e.g., Random Forest), like RAPIDS cuML would be required, which are for NVIDIA GPU acceleration.
4. **Evaluation and Visualization**
   - Training Times for the sequential as well as parallel configurations were captured and compared.
   - Performance Metrics (MAE, MSE, RMSE, R^2 Score) were computed.
   - A side-by-side visualization of outcome (computation time vs. performance) was obtained, validating that parallel processing significantly minimized computation time without sacrificing model accuracy.

Evaluation for Linear Regression:

```
Linear Regression Performance (Parallel):
MAE: 4.7270062910290305
MSE: 38.125269880787506
RMSE: 6.1745663718829285
R2 Score: 0.7549565415424608
```

- MAE = 4.72 : Essentially, on average the model's predictions are off by 4.72 units.
- RMSE: An RMSE of 6.17 indicates the model's predictions were 6.17 units away from the actual value on average. A lower RMSE indicates the model is more accurate.
- R2: This suggests that the model explains roughly 75.4% of the variance in area burned or FRP.

Evaluation for Random Forest Regressor:

```
Random Forest Performance (Parallel):
MAE: 2.822545196205912
MSE: 16.73594800292295
RMSE: 4.090959301059221
R2 Score: 0.8924326413419457
```

- MAE: This indicates that, on average, the model predictions are off by 2.82 units.
- RMSE: This indicates that, on average, the prediction errors are 4.09 units away from the true values, and since it's squared, bigger errors are more severely penalized.
- R2: This indicates that 89.2% of the variance in the target variable is accounted for by the model.

Training time calculation:

```python
import time

# Record training time for Linear Regression (sequential)
start_time = time.time()
lr = LinearRegression()
lr.fit(X_train, y_train)
end_time = time.time()
lr_train_time_sequential = end_time - start_time

# Record training time for Random Forest (sequential)
start_time = time.time()
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
end_time = time.time()
rf_train_time_sequential = end_time - start_time

# Record training time for Linear Regression (parallel fit is not supported, using sequential fit time)
# As noted before, scikit-learn's LinearRegression fit is not parallelizable with n_jobs.
# The fit time will be the same as the sequential version.
lr_train_time_parallel = lr_train_time_sequential

# Record training time for Random Forest (parallel)
start_time = time.time()
rf_parallel = RandomForestRegressor(n_estimators=100, random_state=42, n_jobs=n_cores)
rf_parallel.fit(X_train, y_train)
end_time = time.time()
rf_train_time_parallel = end_time - start_time

# Print the training times
print(f"Linear Regression Training Time (Sequential): {lr_train_time_sequential:.4f} seconds")
print(f"Linear Regression Training Time (Parallel): {lr_train_time_parallel:.4f} seconds (Fit is sequential)")
print(f"Random Forest Training Time (Sequential): {rf_train_time_sequential:.4f} seconds")
print(f"Random Forest Training Time (Parallel): {rf_train_time_parallel:.4f} seconds")
```

```
Linear Regression Training Time (Sequential): 0.0271 seconds
Linear Regression Training Time (Parallel): 0.0271 seconds (Fit is sequential)
Random Forest Training Time (Sequential): 24.8895 seconds
Random Forest Training Time (Parallel): 20.3101 seconds
```

Overview of comparison

```python
# Create a dictionary to store the results
results = {
    'Model': ['Linear Regression', 'Linear Regression', 'Random Forest', 'Random Forest'],
    'Processing': ['Sequential', 'Parallel', 'Sequential', 'Parallel'],
    'MAE': [mae_lr_sequential, mae_lr_parallel, mae_rf_sequential, mae_rf_parallel],
    'MSE': [mse_lr_sequential, mse_lr_parallel, mse_rf_sequential, mse_rf_parallel],
    'RMSE': [rmse_lr_sequential, rmse_lr_parallel, rmse_rf_sequential, rmse_rf_parallel],
    'R2 Score': [r2_lr_sequential, r2_lr_parallel, r2_rf_sequential, r2_rf_parallel],
    'Training Time (s)': [lr_train_time_sequential, lr_train_time_parallel, rf_train_time_sequential, rf_train_time_parallel]
}

# Create a pandas DataFrame from the dictionary
results_df = pd.DataFrame(results)

# Display the DataFrame
display(results_df)
```

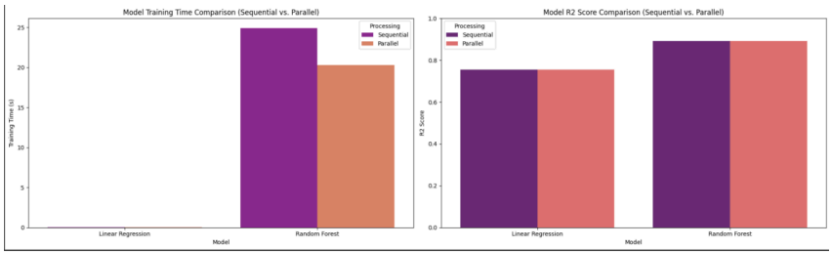
	Model	Processing	MAE	MSE	RMSE	R2 Score	Training Time (s)
0	Linear Regression	Sequential	4.727006	38.125270	6.174566	0.754957	0.027135
1	Linear Regression	Parallel	4.727006	38.125270	6.174566	0.754957	0.027135
2	Random Forest	Sequential	2.822545	16.735948	4.090959	0.892433	24.889498
3	Random Forest	Parallel	2.822545	16.735948	4.090959	0.892433	20.310144

Visualization

```python
# Plot Training Time Comparison
plt.figure(figsize=(20, 6)) # Increase figure size for side-by-side plots
plt.subplot(1, 2, 1) # First plot on the left
sns.barplot(x='Model', y='Training Time (s)', hue='Processing', data=results_df, palette='plasma')
plt.title('Model Training Time Comparison (Sequential vs. Parallel)')
plt.ylabel('Training Time (s)')
plt.xlabel('Model')

# Plot R2 Score Comparison
plt.subplot(1, 2, 2) # Second plot on the right
sns.barplot(x='Model', y='R2 Score', hue='Processing', data=results_df, palette='magma')
plt.title('Model R2 Score Comparison (Sequential vs. Parallel)')
plt.ylabel('R2 Score')
plt.xlabel('Model')
plt.ylim(0, 1) # R2 Score is between 0 and 1

plt.tight_layout() # Adjust layout to prevent labels overlapping
plt.show()
```



4. Ethical and Professional Responsibility

Fairness and Bias: The model may be biased toward historical trends in the historical database and may miss new fire trends due to climate change, or underreported regions.

Impact: Misallocated fire response resources (e.g. spending more resources on a seasonal risk), ignoring vulnerable regions, and bias towards particular geography.

Mitigation:

- Utilising recent andviable data from regions and seasons.
- Train the model repeatedly on novel incident data.
- Conduct fairness audits on the model performance from heterogeneous regions and time frames

False Positives / Negatives:

- **False Positives:** Reporting a fire where there will not be a fire.
  Cost to the fire community: Time and resources wasted, panicking the public.
- **False Negatives**: Not reporting the actual fire.
  Cost to the fire community: Potential loss of life, property, and natural space.

Minimizing False Positives and False Negatives:

- Setting thresholds using tolerance management.
- Reporting confidence intervals, uncertainty quantification.
- Use human judgement/oversight, i.e. validation of high-risk alerts by an expert.

User Privacy

When used with satellite, drone, or geographical monitoring, system may capture private information without user consent.

Possible mitigation strategies:

- Implement privacy-preserving methodologies, such as: differential privacy, de-identified geo data.

System Failure

System may crash or prediction failures may happen during peak fire season.

Mitigation

- Design for redundancy and failover.
- Test regularly with back-up strategies.
- Develop alerts and monitoring pipelines to track real-time performance.

5. System Architecture and Deployment Planning

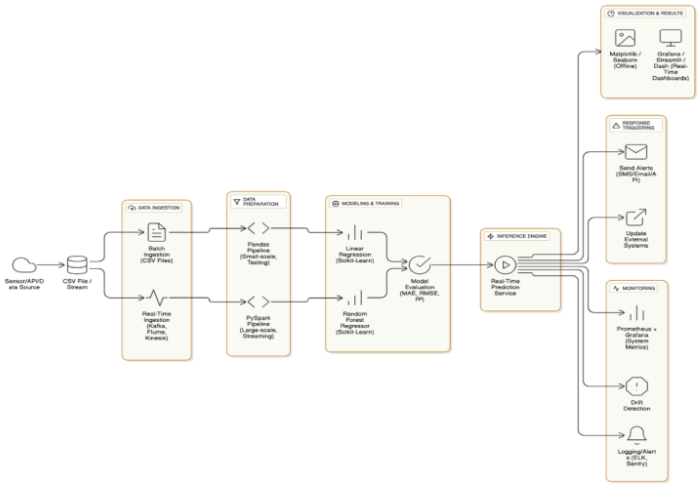


Figure 19: System Architecture (Eraser, n.d.)

Data Ingestion

Component: Data Source CSV

Real-Time Extension: The real-time extension could ingest from a real-time data source such as a streaming API, e.g., Kafka, Flume, or Kinesis, instead of CSV files.

Use: To continuously ingest data from sensors, logs, or real-time source data into the platform.

Data Preparation

Component: Pipelines for PySpark and Pandas

Function:

- PySpark: Scalable distributed data processing for large-scale streaming input,.
- Pandas: Smaller batch processing or rapid prototyping/testing.

Real-Time Implementation: Spark Streaming or Dask can be set up to ingest streaming data and process/transformation in real-time.

Modeling

Component: Linear Regression and Random Forest Regressor with Scikit-Learn

Function: The models are trained and serialized offline. Then drawn in at inference time.

Model Training & Evaluation

Component:

- Train Models (offline trained, retrained regularly)
- Evaluate with MSE and R2
- Predict on new data

Real-Time: The inference is done in real time with the latest trained models. Evaluation metrics are used in logs or dashboards after the model is deployed.

Results

Component:

- Compare Models and Aggregation Time: Good for benchmarking purposes
- Visualize Results: Real-time dashboards in Matplotlib or Seaborn (or production-time dashboards with Plotly Dash, Grafana, or Streamlit)

Real-Time Output: The modules trigger alerts or visualizations that allow the stakeholders or downstream systems to be informed in real time.

Response Triggering

Extension not explicit in diagram (but implied):

- Follow up events based on model output (e.g.):
  - Alert notifications (email/SMS)
  - API callbacks
  - Updating external systems (Chatgpt, n.d.)

Monitoring (Professional Reliability)

Real-time additions:

- For system metrics, use Prometheus + Grafana
- Have model drift detectors to monitor the input distribution
- Create logging and alerting pipelines (e.g. ELK stack, Sentry) (Chatgpt, n.d.)

6. Professional Communication

6.1 Self-Reflection

It has been a completely enriching experience to work on the wildfire forecast project. I was able to go deep into actual-world environmental data drawn from Kaggle and learn how machine learning can effectively be used for climate-based hazard forecasting. The dataset, consisting of more than two decades' worth of records of fire occurrences in Turkey, gave me a good chance to realize the significance of data preprocessing, feature engineering, and model evaluation within the domain of environmental science.

One of my favorite moments was to apply both Linear Regression and Random Forest models for the forecasting of fire severity. I learned how to properly split data, train models, and test them

using evaluation metrics. I enjoyed the manner Random Forest, with parallel computation, significantly reduced training time and improved model performance. Learning parallel computing further deepened my knowledge on how extensive sets of information can be processed effectively with multiprocessing techniques.

Besides the technical training, this project also enhanced my problem-solving abilities and analytical mind. It also sensitized me to the significance of data in informing disaster management policies and climate change research. Overall, this project enhanced my practical machine learning skills as well as sensitization on how technology can be utilized in solving real-world environmental problems.

6.2 Documentation and Transparency

- In-depth documentation was kept throughout. This included:
- Annotated code cells explaining logic for preprocessing, model selection, and evaluation.
- Consistent naming conventions for variables and functions.
- Comments indicating assumptions, decisions taken, and reasoning for model choice (e.g., selection of Linear Regression and Random Forest based on interpretability and performance).
- Proper citation of external data sources (e.g., Kaggle wildfire dataset) and third-party code or reference used in the report.

Following is GitHub link to the file having code:

https://github.com/Sushant2080-0140/MLPC

References

(n.d.). Retrieved from Eraser: https://app.eraser.io/

Apache Spark. (n.d.). Retrieved from databricks: https://www.databricks.com/glossary/what-is-apache-spark

Apache spark: A unified engine for big data processing. (n.d.). *59*. doi:10.1145/2934664

Chatgpt. (n.d.). Retrieved from https://chatgpt.com/

Dincer, B. (2021, August 10). *Kaggle*. Retrieved from https://www.kaggle.com/datasets/brsdincer/20002021-turkey-fire-points-single-csv-nasa/data

ForestFireManagement. (n.d.). Retrieved from https://www.fao.org/forestry/firemanagement/en/

Guolin Ke, Q. M.-Y. (n.d.). LightGBM: A Highly Efficient Gradient Boosting Decision Tree. 9. Retrieved from https://papers.nips.cc/paper_files/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf

IPCC Intergovernmental Panel on Climate Change. (n.d.). Retrieved from Climate change 2021: The Physical Science Basis: https://www.ipcc.ch/report/ar6/wg1/

Kumar, A. (2023, November 13). *R-squared in Linear Regression Models: Concepts, Examples*. Retrieved from Analytics Yogi: https://vitalflux.com/r-squared-explained-machine-learning/

Linear regression in machine learning. (2025, July 11). Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/machine-learning/ml-linear-regression/

MapReduce: simplified data processing on large clusters. (n.d.). *51*. doi:10.1145/1327452.1327492

Multiprocessing - process-based parallelism. (n.d.). Retrieved from Python documentation: https://docs.python.org/3/library/multiprocessing.html

nfpa.org. (n.d.). Retrieved from https://www.nfpa.org/education-and-research/wildfire

Random Forest Regression in Python. (2025, July 12). Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/machine-learning/random-forest-regression-in-python/

Remote Sensing of Environment. (n.d.). (Elsevier) Retrieved from https://www.sciencedirect.com/science/article/pii/S0034425716300827?via%3Dihub

What is Root Mean Square Error? (n.d.). Retrieved from c3.ai: https://c3.ai/glossary/data-science/root-mean-square-error-rmse/

ITS66604_0369922_INDVASGNMT

ORIGINALITY REPORT

13%	10%	6%	7%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	www.geeksforgeeks.org Internet Source	1%
2	upcommons.upc.edu Internet Source	1%
3	Federica Gazzelloni. "Health Metrics and the Spread of Infectious Diseases - Machine Learning Applications and Spatial Modeling Analysis with R", CRC Press, 2025 Publication	1%
4	Submitted to Taylor's Education Group Student Paper	1%
5	Submitted to University of Wales Swansea Student Paper	1%
6	ijirt.org Internet Source	1%
7	aihubprojects.com Internet Source	1%
8	github.com Internet Source	1%
9	Matei Zaharia, Michael J. Franklin, Ali Ghodsi, Joseph Gonzalez et al. "Apache Spark", Communications of the ACM, 2016 Publication	<1%
10	T. Mariprasath, Kumar Reddy Cheepati, Marco Rivera. "Practical Guide to Machine Learning, NLP, and Generative AI: Libraries, Algorithms, and Applications", River Publishers, 2024 Publication	<1%

11	Submitted to Brisbane Catholic Education Student Paper	<1%
12	H L Gururaj, Francesco Flammini, V Ravi Kumar, N S Prema. "Recent Trends in Healthcare Innovation", CRC Press, 2025 Publication	<1%
13	Submitted to Westcliff University Student Paper	<1%
14	Submitted to Springs Charter Schools Student Paper	<1%
15	fastercapital.com Internet Source	<1%
16	digital.library.adelaide.edu.au Internet Source	<1%
17	ntnuopen.ntnu.no Internet Source	<1%
18	arxiv.org Internet Source	<1%
19	Submitted to cit Student Paper	<1%
20	ijaim.net Internet Source	<1%
21	iieta.org Internet Source	<1%
22	issuu.com Internet Source	<1%
23	www.coursehero.com Internet Source	<1%

Exclude quotes	On	Exclude matches	Off
Exclude bibliography	On		