Innovative Approaches in Material Science: Leveraging Machine Learning for Enhanced Polymeric Nanocomposites to Predict Tensile Strength of CNT/Composites in Elastomer Nano-Composites

A Project Report

Master of Technology in Chemical Engineering

Submitted by
Murari Singh
Entry No. 22CH60R65

DEPARTMENT OF CHEMICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR
WEST BENGAL - 721302, INDIA

November 25, 2023

Innovative Approaches in Material Science: Leveraging Machine Learning for Enhanced Polymeric Nanocomposites

November 26, 2023

Contents

Innovative Approaches in Material Science: Leveraging Machine Learning for Enhanced Polymeric Nanocomposites

Introduction

In the rapidly evolving landscape of material science, the urgency to accelerate the development of new materials has never been more critical. This study zeroes in on the burgeoning field of **polymeric nanocomposites**, particularly those reinforced with carbon nanotubes (CNTs), where traditional experimental methods fall short due to their prolonged duration and cost-intensive nature.

Global initiatives like the Materials Genome Initiative (MGI), introduced in the USA by President Obama in 2011, are reshaping the landscape, aiming to halve the development time and cost of new materials. The initiative demonstrates the power of integrating computational tools, experimental resources, and digital data in revolutionizing material development.

The focus of this study is *carbon nanotubes (CNTs)*, whose exceptional properties make them invaluable in enhancing polymer matrices. Yet, challenges in their distribution and surface modification within the polymer matrix present significant hurdles.

The cornerstone of our research is the development of an advanced computational tool using the **Random Forest Regressor Model**, designed to predict the tensile strength of CNT/polymer nanocomposites. This model is trained and validated against a dataset comprising 96 configurations and 12 different variables, harnessing the potential of machine learning. The identified input features include the density of the polymer, Young's modulus of the matrix, tensile strength of the matrix, CNT weight fraction (%), CNT density, CNT average diameter, CNT average length, Young's modulus of CNT, polymer matrix, CNT surface modification method, and nanocomposite processing method.

Through this pioneering approach, we aim to deeply understand the effective properties of CNT nanocomposites, opening new avenues in material science and engineering, and paving the way for groundbreaking advancements.

Structure of the Paper: The paper is structured as follows: Section 2 discusses the Database, Section 3 focuses on Model Building, Section 4 delves into Model Evaluation, Section 5 presents the Results, Section 6 concludes with the Conclusion, Section 7 outlines Future Work, and Section 8 lists the References.

1 Data Base

1.1 Dataset

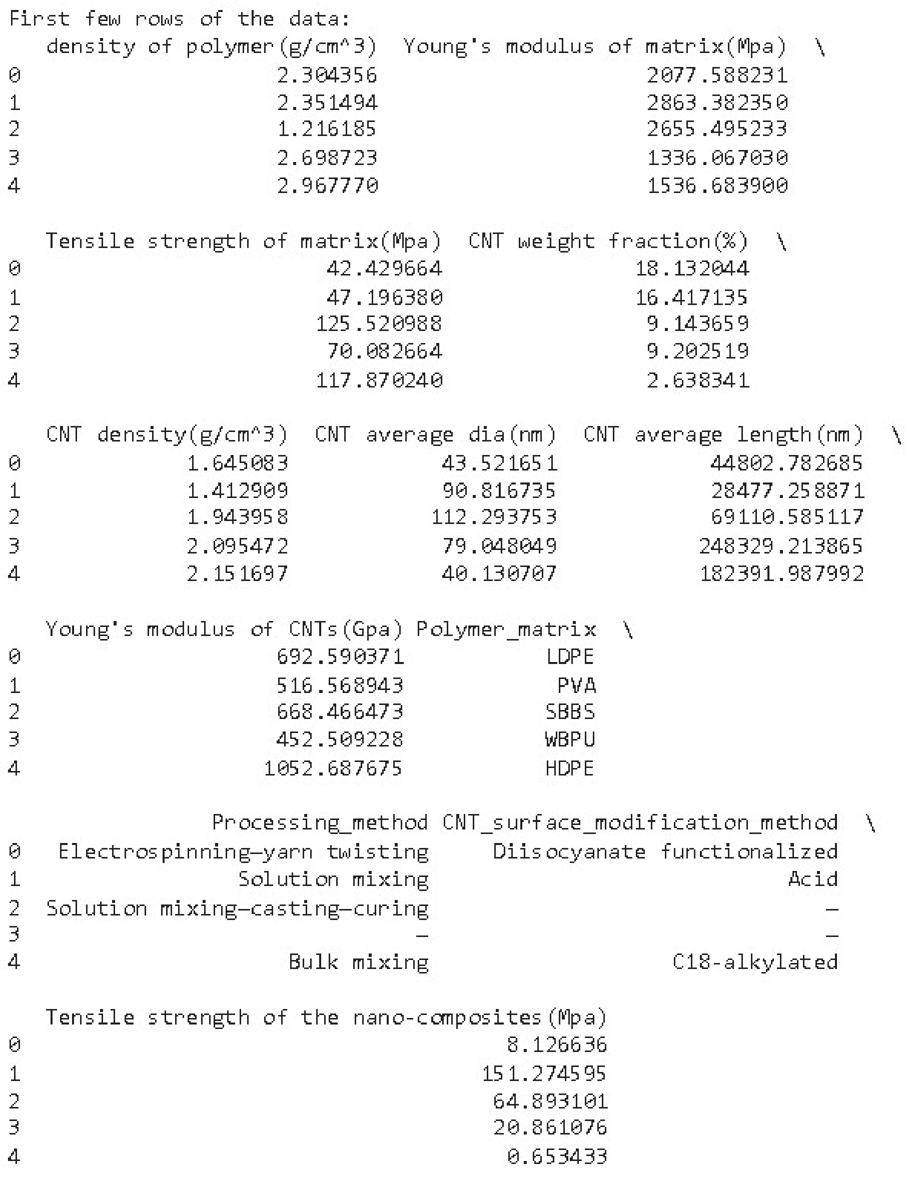
The dataset used for this analysis is *Combined_Polymer_Dataset__murari_singh.xlsx*. It consists of both numerical and categorical features.

1.2 Steps and Analysis

1. **Data Loading and Preliminary Analysis**
   - Loaded the dataset using pandas.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# File path for the Excel file
file_path = 'Combined_Polymer_Dataset__murari_singh.xlsx'

# Read the first few lines of the Excel file
data = pd.read_excel(file_path)
print("First␣few␣rows␣of␣the␣data:")
print(data.head()) # Displays the first 5 rows
```

   - Performed initial exploration to understand the data structure.
   - Checked for missing values and found none.



```
First few rows of the data:
   density of polymer(g/cm^3)  Young's modulus of matrix(Mpa)  \
0                    2.304356                     2077.588231
1                    2.351494                     2863.382350
2                    1.216185                     2655.495233
3                    2.698723                     1336.067030
4                    2.967770                     1536.683900

   Tensile strength of matrix(Mpa)  CNT weight fraction(%)  \
0                        42.429664               18.132044
1                        47.196380               16.417135
2                       125.520988                9.143659
3                        70.082664                9.202519
4                       117.870240                2.638341

   CNT density(g/cm^3)  CNT average dia(nm)  CNT average length(nm)  \
0             1.645083            43.521651            44802.782685
1             1.412909            90.816735            28477.258871
2             1.943958           112.293753            69110.585117
3             2.095472            79.048049           248329.213865
4             2.151697            40.130707           182391.987992

   Young's modulus of CNTs(Gpa) Polymer_matrix  \
0                    692.590371           LDPE
1                    516.568943            PVA
2                    668.466473           SBBS
3                    452.509228           WBPU
4                   1052.687675           HDPE

                 Processing_method CNT_surface_modification_method  \
0   Electrospinning-yarn twisting     Diisocyanate functionalized
1                Solution mixing                            Acid
2  Solution mixing-casting-curing                               -
3                               -                               -
4                     Bulk mixing                   C18-alkylated

   Tensile strength of the nano-composites(Mpa)
0                                      8.126636
1                                    151.274595
2                                     64.893101
3                                     20.861076
4                                      0.653433
```

Figure 1: First few rows of the data.

```
missing_values = data.isnull().sum()
print("\nMissing␣values␣in␣each␣column:")
print(missing_values)
```

Performed initial exploration to understand the data structure.

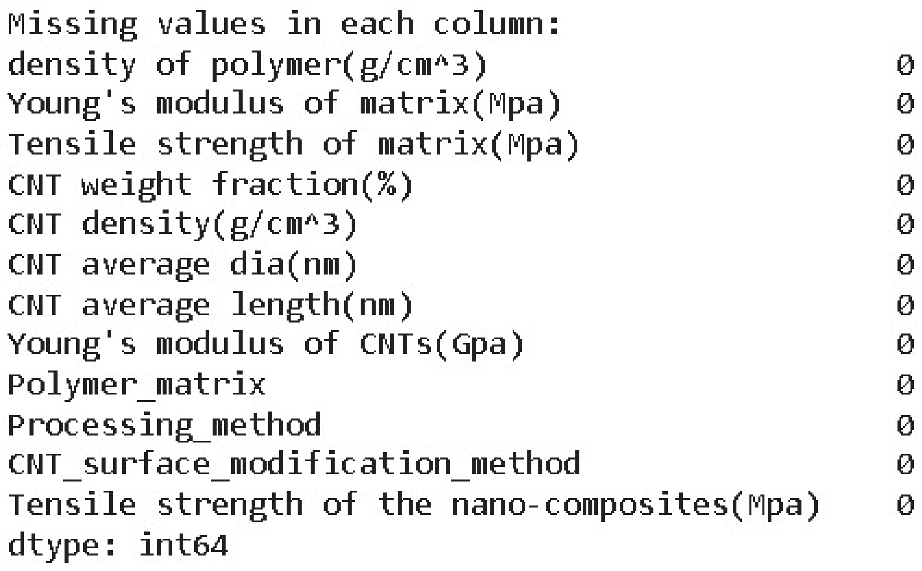
Checked for missing values and found none.

Generated summary statistics to understand data distribution.

Created box plots for numerical columns to visualize their distributions and identify outliers.

Generated summary statistics to understand data distribution.

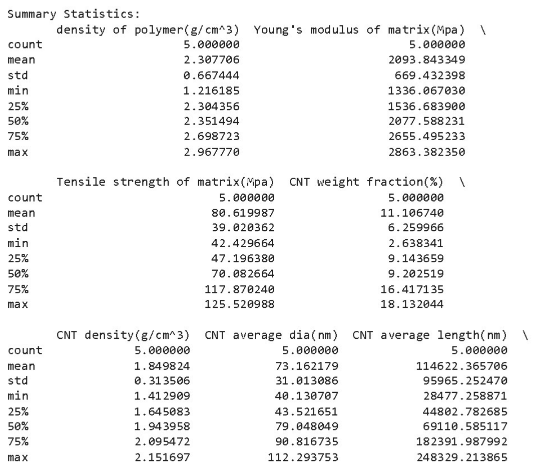
```
# Summary statistics of the data
print("\nSummary␣Statistics:")
print(data.describe())

# Creating box plots for all numerical columns
plt.figure(figsize=(15, 10))
```



```
Missing values in each column:
density of polymer(g/cm^3)                            0
Young's modulus of matrix(Mpa)                        0
Tensile strength of matrix(Mpa)                       0
CNT weight fraction(%)                                0
CNT density(g/cm^3)                                   0
CNT average dia(nm)                                   0
CNT average length(nm)                                0
Young's modulus of CNTs(Gpa)                          0
Polymer_matrix                                        0
Processing_method                                     0
CNT_surface_modification_method                       0
Tensile strength of the nano-composites(Mpa)          0
dtype: int64
```

Figure 2: Missing values in each column.

```
sns.boxplot(data=data.select_dtypes(include=['float64', 'int64']))
plt.xticks(rotation=45)
plt.title('Box␣plot␣of␣Numerical␣Columns')
plt.show()
```



```
Summary Statistics:
       density of polymer(g/cm^3)  Young's modulus of matrix(Mpa)  \
count                    5.000000                        5.000000
mean                     2.307706                     2093.843349
std                      0.667444                      669.432398
min                      1.216185                     1336.067030
25%                      2.304356                     1536.683900
50%                      2.351494                     2077.588231
75%                      2.698723                     2655.495233
max                      2.967770                     2863.382350

       Tensile strength of matrix(Mpa)  CNT weight fraction(%)  \
count                         5.000000                5.000000
mean                         80.619987               11.106740
std                          39.020362                6.259966
min                          42.429664                2.638341
25%                          47.196380                9.143659
50%                          70.082664                9.202519
75%                         117.870240               16.417135
max                         125.520988               18.132044

       CNT density(g/cm^3)  CNT average dia(nm)  CNT average length(nm)  \
count             5.000000             5.000000                5.000000
mean              1.849824            73.162179           114622.365706
std               0.313506            31.013086            95965.252470
min               1.412909            40.130707            28477.258871
25%               1.645083            43.521651            44802.782685
50%               1.943958            79.048049            69110.585117
75%               2.095472            90.816735           182391.987992
max               2.151697           112.293753           248329.213865
```

Figure 3: Summary statistics of the dataset - Part 1.

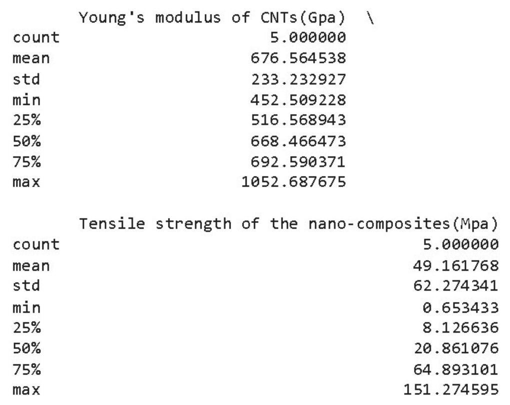
1.3 Data Preprocessing

- *Normalization:* Numerical features were normalized using StandardScaler. This step is crucial for algorithms like Random Forest that are sensitive to the scale of input features.

Mathematical Description of Normalization

The most common methods of normalization are Min-Max Scaling and Standardization (Z-score normalization).

1. Min-Max Scaling
   Min-Max Scaling scales the features to a given range, usually 0 to 1. The formula for Min-Max Scaling



```
       Young's modulus of CNTs(Gpa)  \
count                      5.000000
mean                     676.564538
std                      233.232927
min                      452.509228
25%                      516.568943
50%                      668.466473
75%                      692.590371
max                     1052.687675

       Tensile strength of the nano-composites(Mpa)
count                                      5.000000
mean                                      49.161768
std                                       62.274341
min                                        0.653433
25%                                        8.126636
50%                                       20.861076
75%                                       64.893101
max                                      151.274595
```

Figure 4: Summary statistics of the dataset - Part 2.

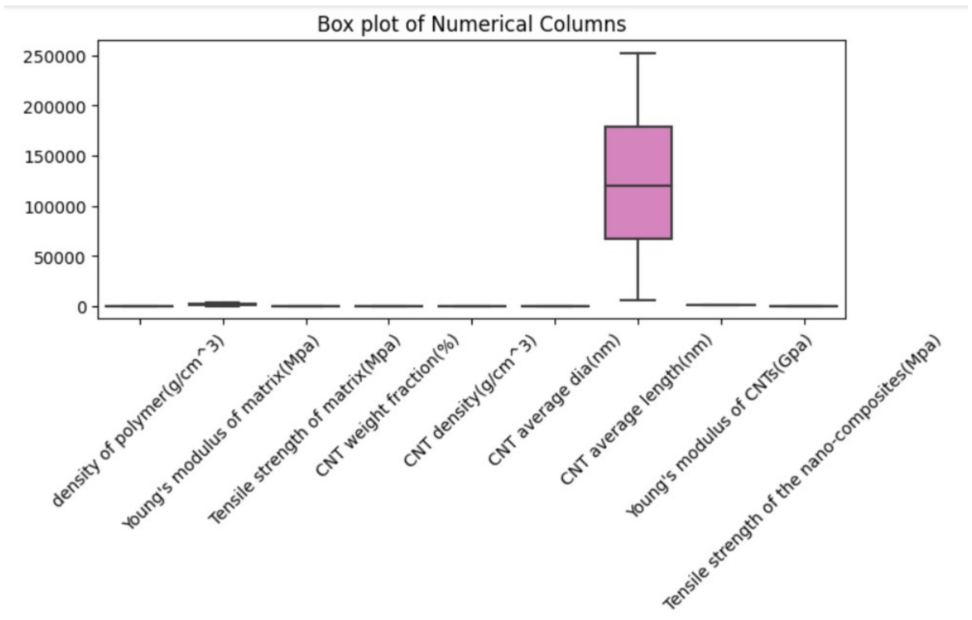


Figure 5: Box plot of Numerical Columns.

is:

$$X_{\text{scaled}} = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}}$$

where:

- X is the original value.
- X_{min} is the minimum value of the feature.
- X_{max} is the maximum value of the feature.
- X_{scaled} is the scaled value.

2. Standardization (Z-score Normalization)

   Standardization transforms the data to have a mean of zero and a standard deviation of one. The formula for Standardization is:

$$Z = \frac{X - \mu}{\sigma}$$

where:

- X is the original value.
- μ is the mean of the feature.
- σ is the standard deviation of the feature.
- Z is the standardized value (also called the Z-score).

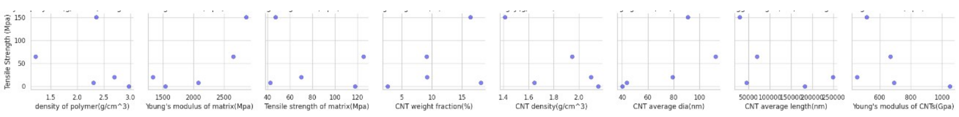


Figure 6: Scatter Plot Against Each Input.

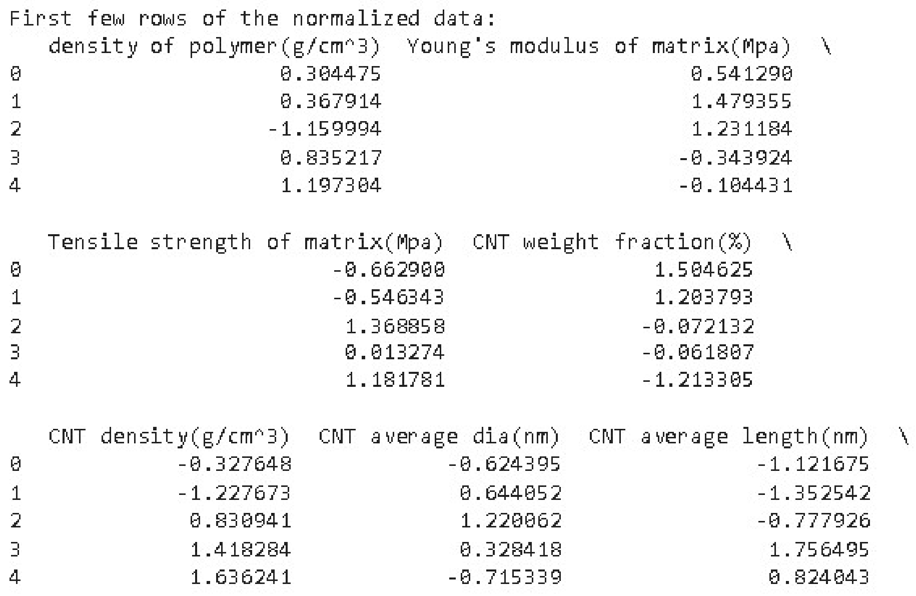
Importance of Normalization

- Equal Contribution: Ensures that each feature contributes equally to the distance computations.
- Faster Convergence: In gradient descent algorithms, normalization can result in faster convergence.
- Handling Skewed Data: Helps in managing features with different units and scales.

When to Use

- Standardization is generally recommended when the features follow a normal distribution.
- Min-Max Scaling is often chosen for cases where the algorithm requires data to be bounded within a specific range (e.g., neural networks).

```
import pandas as pd
from sklearn.preprocessing import StandardScaler

# Separating numerical and categorical columns
numerical_cols = X.select_dtypes(include=['float64', 'int64']).columns
categorical_cols = X.select_dtypes(include=['object']).columns

# Normalizing the numerical columns
scaler = StandardScaler()
X_normalized = X.copy()
X_normalized[numerical_cols] = scaler.fit_transform(X[numerical_cols])

print("First␣few␣rows␣of␣the␣normalized␣data:")
print(X_normalized.head())
```

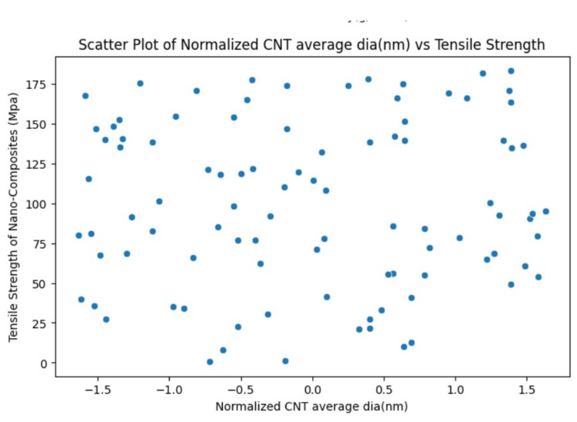


```
First few rows of the normalized data:
   density of polymer(g/cm^3)  Young's modulus of matrix(Mpa)  \
0                    0.304475                        0.541290   
1                    0.367914                        1.479355   
2                   -1.159994                        1.231184   
3                    0.835217                       -0.343924   
4                    1.197304                       -0.104431   

   Tensile strength of matrix(Mpa)  CNT weight fraction(%)  \
0                        -0.662900                1.504625   
1                        -0.546343                1.203793   
2                         1.368858               -0.072132   
3                         0.013274               -0.061807   
4                         1.181781               -1.213305   

   CNT density(g/cm^3)  CNT average dia(nm)  CNT average length(nm)  \
0            -0.327648            -0.624395               -1.121675   
1            -1.227673             0.644052               -1.352542   
2             0.830941             1.220062               -0.777926   
3             1.418284             0.328418                1.756495   
4             1.636241            -0.715339                0.824043   
```

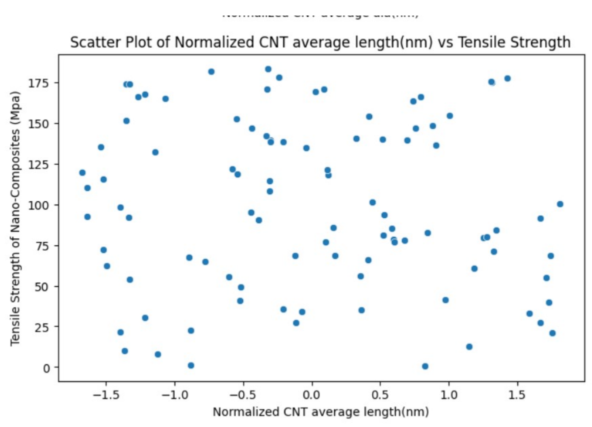
Figure 7: First few rows of the data after normalization.

One-Hot Encoding

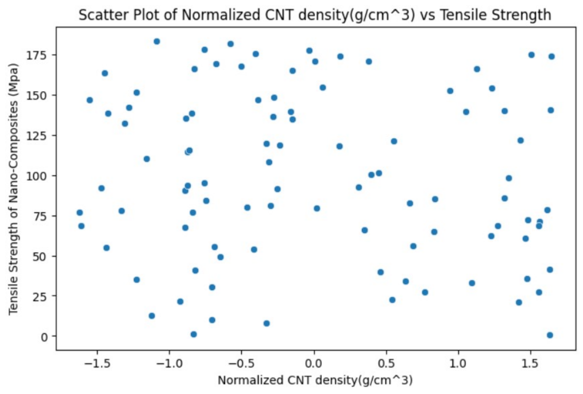
One-Hot Encoding is a technique used to convert categorical data into a numerical format, making it suitable for many types of machine learning algorithms that require numerical input. It involves representing each



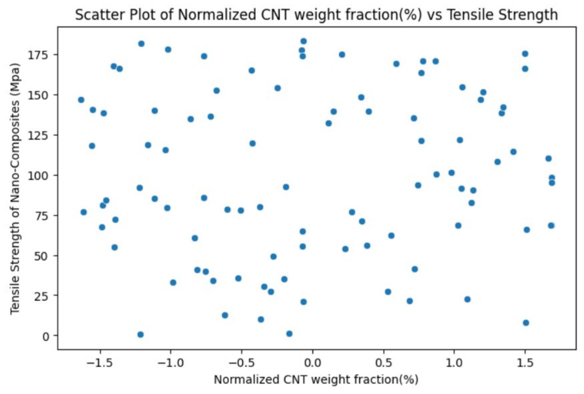
(a) CNT Average Diameter



(b) CNT Average Length



(c) CNT Density



(d) CNT Weight Fraction

Figure 8: Various normalized data graphs

categorical variable with a binary vector.

Mathematical Description of One-Hot Encoding Suppose you have a categorical feature with N distinct categories (or classes). One-Hot Encoding transforms this feature into N binary columns, where each column corresponds to one of the categories. In each of these columns, the presence of a category in the original data is marked with a 1, and the absence is marked with a 0.

Example Consider a categorical feature ”Color” with three categories: Red, Green, and Blue.

Color_Red	Color_Green	Color_Blue
1	0	0
0	1	0
0	0	1
1	0	0

Table 1: One-hot encoded representation of the feature ”Color”.

In the encoded data:
- ”Color_Red” is 1 when the original color is Red and 0 otherwise.
- ”Color_Green” is 1 when the original color is Green and 0 otherwise.
- ”Color_Blue” is 1 when the original color is Blue and 0 otherwise.

Mathematical Representation For a categorical variable X with N unique categories, one-hot encoding can be represented as a function $f(X) \to R^N$, where R^N is an N-dimensional binary vector.

The encoding function f can be defined as:

$$f(X = \text{category}_i) = [0, 0, ..., 1, ..., 0, 0]$$

where the 1 is at the i-th position in the vector, and all other positions are 0.

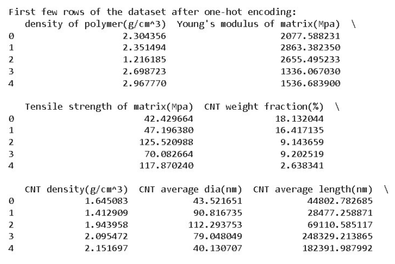
Importance in Machine Learning

- Compatibility with Algorithms: Many machine learning models, especially those based on numerical computations, require numerical input. One-Hot Encoding allows these models to process categorical data.
- Avoiding False Numerical Relationships: It prevents models from assuming a natural ordering among categories, which might be misleading (e.g., treating the category ”Red” as less than ”Green” if encoded as 1 ¡ 2).

Considerations

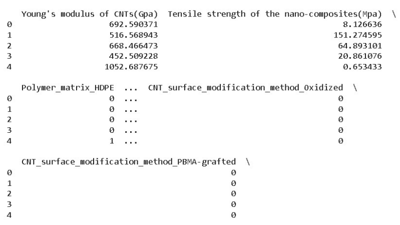
- Dimensionality: One-Hot Encoding can significantly increase the dataset’s dimensionality (number of features), leading to issues like the curse of dimensionality, especially with high cardinality features.
- Sparse Representation: The encoded matrix is often sparse, which might require more storage and computational resources.

```
import pandas as pd
from sklearn.preprocessing import OneHotEncoder

# One-hot encoding the categorical variables
X_encoded = pd.get_dummies(data, columns=categorical_cols)

# Displaying the first few rows of the dataset after encoding
print("First␣few␣rows␣of␣the␣dataset␣after␣one-hot␣encoding:")
print(X_encoded.head())
```



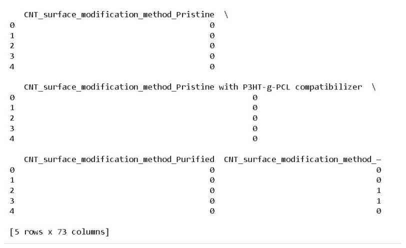
```
First few rows of the dataset after one-hot encoding:
   density of polymer(g/cm^3)  Young's modulus of matrix(Mpa)  \
0                    2.304356                     2077.588231
1                    2.351494                     2863.382350
2                    1.216185                     2655.495233
3                    2.698723                     1336.067030
4                    2.967770                     1536.683900

   Tensile strength of matrix(Mpa)  CNT weight fraction(%)  \
0                        42.429664               18.132044
1                        47.196380               16.417135
2                       125.520988                9.143659
3                        70.082664                9.202519
4                       117.870240                2.638341

   CNT density(g/cm^3)  CNT average dia(nm)  CNT average length(nm)  \
0             1.645083            43.521651            44802.782685
1             1.412909            90.816735            28477.258871
2             1.943958           112.293753            69110.585117
3             2.095472            79.048049           248329.213865
4             2.151697            40.130707           182391.987992
```

(a) Part 1



```
   Young's modulus of CNTs(Gpa)  Tensile strength of the nano-composites(Mpa)  \
0                    692.590371                                      8.126636
1                    516.568943                                    151.274595
2                    668.466473                                     64.893101
3                    452.509228                                     20.861076
4                   1052.687675                                      0.653433

   Polymer_matrix_HDPE  ...  CNT_surface_modification_method_Oxidized  \
0                    0  ...                                         0
1                    0  ...                                         0
2                    0  ...                                         0
3                    0  ...                                         0
4                    1  ...                                         0

   CNT_surface_modification_method_PBMA-grafted  \
0                                             0
1                                             0
2                                             0
3                                             0
4                                             0
```

(b) Part 2



```
   CNT_surface_modification_method_Pristine  \
0                                         0
1                                         0
2                                         0
3                                         0
4                                         0

   CNT_surface_modification_method_Pristine with P3HT-g-PCL compatibilizer  \
0                                                  0
1                                                  0
2                                                  0
3                                                  0
4                                                  0

   CNT_surface_modification_method_Purified  CNT_surface_modification_method_-
0                                         0                                  0
1                                         0                                  0
2                                         0                                  1
3                                         0                                  1
4                                         0                                  0

[5 rows x 73 columns]
```

(c) Part 3

Figure 9: First few rows of the dataset after one-hot encoding

- Separated the dataset into input features (X) and the target variable (Y). The target variable is ”Tensile strength of the nano-composites (MPa)”.

Separating Input Features and Target Variable

The Python code below is used to separate the dataset into input features (X) and the target variable (Y):

```
import numpy as np

# Converting the dataframe to a numpy array
dataset = data.to_numpy()

# Dropping the target variable to separate features
X = data.drop('Tensile␣strength␣of␣the␣nano-composites(Mpa)', axis=1) #
    red↪ Input features
Y = data['Tensile␣strength␣of␣the␣nano-composites(Mpa)'] # Target
    red↪ variable
```

After executing the above code, the shape of the dataset, as well as the separated input features (X) and target variable (Y), are as follows:

- Shape of the entire dataset: $(96, 12)$
- Shape of X (features): $(96, 11)$
- Shape of Y (target): $(96,)$

The first five entries of the target variable (Y) are:

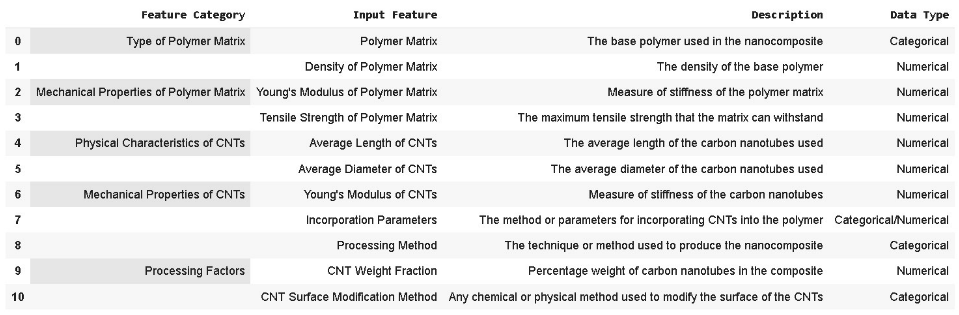
```
0      8.126636
1    151.274595
2     64.893101
3     20.861076
4      0.653433
Name: Tensile strength of the nano-composites(Mpa), dtype: float64
```

The output indicates the shape of the entire dataset, as well as the separated input features (X) and the target variable (Y). The first few entries of the target variable are also displayed for verification.

2 Model Building

2.1 All the Input Features

This subsection discusses all the input features used in the model. It includes details on the data used for training the Random Forest model, such as the physical and chemical properties of the materials, their processing methods, and other relevant features.



	Feature Category	Input Feature	Description	Data Type
0	Type of Polymer Matrix	Polymer Matrix	The base polymer used in the nanocomposite	Categorical
1		Density of Polymer Matrix	The density of the base polymer	Numerical
2	Mechanical Properties of Polymer Matrix	Young's Modulus of Polymer Matrix	Measure of stiffness of the polymer matrix	Numerical
3		Tensile Strength of Polymer Matrix	The maximum tensile strength that the matrix can withstand	Numerical
4	Physical Characteristics of CNTs	Average Length of CNTs	The average length of the carbon nanotubes used	Numerical
5		Average Diameter of CNTs	The average diameter of the carbon nanotubes used	Numerical
6	Mechanical Properties of CNTs	Young's Modulus of CNTs	Measure of stiffness of the carbon nanotubes	Numerical
7		Incorporation Parameters	The method or parameters for incorporating CNTs into the polymer	Categorical/Numerical
8		Processing Method	The technique or method used to produce the nanocomposite	Categorical
9	Processing Factors	CNT Weight Fraction	Percentage weight of carbon nanotubes in the composite	Numerical
10		CNT Surface Modification Method	Any chemical or physical method used to modify the surface of the CNTs	Categorical

Figure 10: Overview of the Input Parameters.

2.2 Polymer Matrix, Surface Modification Method, and Processing Method

In this subsection, the focus is on the polymer matrix, surface modification methods, and processing methods used. These factors significantly impact the performance and characteristics of the final composite material.

2.3 The Random Forest Regressor Algorithm

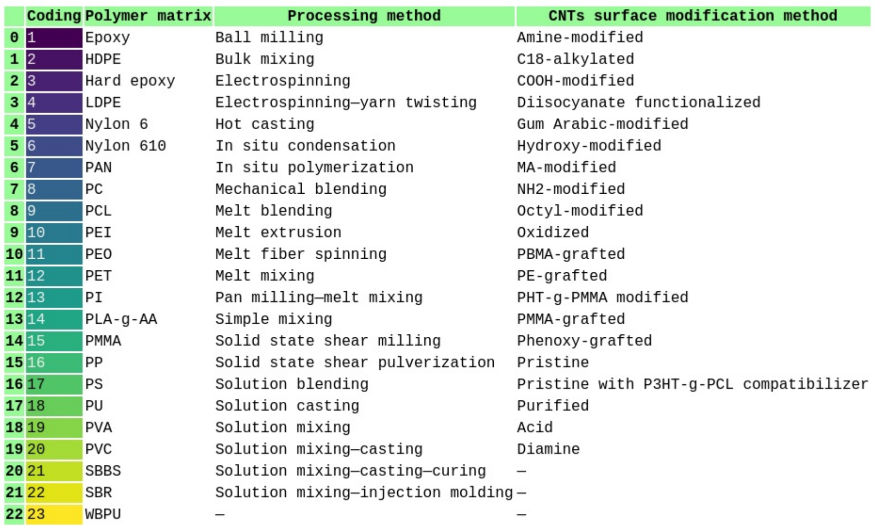
The Random Forest Regressor Algorithm is a powerful ensemble learning method used for regression tasks. It operates by constructing a multitude of decision trees at training time and outputting the average prediction of the individual trees for regression tasks. The underlying concept of a Random Forest lies in the power of 'ensemble learning,' where multiple models combine to solve a single prediction problem, often leading to better results than any single model alone.

Math Description

Mathematical Description of Random Forest Regression

1. **Bootstrapping**

   A Random Forest starts by bootstrapping the dataset; that is, it samples n instances with replacement from the dataset to create a subset (this subset is as large as the original dataset). This process is repeated to create as many subsets as there are trees in the forest.



	Coding	Polymer matrix	Processing method	CNTs surface modification method
0	1	Epoxy	Ball milling	Amine-modified
1	2	HDPE	Bulk mixing	C18-alkylated
2	3	Hard epoxy	Electrospinning	COOH-modified
3	4	LDPE	Electrospinning—yarn twisting	Diisocyanate functionalized
4	5	Nylon 6	Hot casting	Gum Arabic-modified
5	6	Nylon 610	In situ condensation	Hydroxy-modified
6	7	PAN	In situ polymerization	MA-modified
7	8	PC	Mechanical blending	NH2-modified
8	9	PCL	Melt blending	Octyl-modified
9	10	PEI	Melt extrusion	Oxidized
10	11	PEO	Melt fiber spinning	PBMA-grafted
11	12	PET	Melt mixing	PE-grafted
12	13	PI	Pan milling—melt mixing	PHT-g-PMMA modified
13	14	PLA-g-AA	Simple mixing	PMMA-grafted
14	15	PMMA	Solid state shear milling	Phenoxy-grafted
15	16	PP	Solid state shear pulverization	Pristine
16	17	PS	Solution blending	Pristine with P3HT-g-PCL compatibilizer
17	18	PU	Solution casting	Purified
18	19	PVA	Solution mixing	Acid
19	20	PVC	Solution mixing—casting	Diamine
20	21	SBBS	Solution mixing—casting—curing	—
21	22	SBR	Solution mixing—injection molding	—
22	23	WBPU	—	—

Figure 11: Details of All Polymer Matrix.

2. **Tree Building**

   For each bootstrap sample, a decision tree is built. The algorithm makes the following considerations:

   - At each node, instead of searching through all features to find the best feature to split the data, it searches through a random subset of features. The number of features that can be searched at each split is a parameter of the algorithm and is denoted by m. Typically, m is chosen as the square root of the total number of features.
   - Each tree is grown to the largest extent possible, and there is no pruning.

3. **Prediction**

   For a regression problem, the prediction from the Random Forest regressor is made by averaging the predictions of all the individual trees, which is mathematically represented as:

$$Y = \frac{1}{N}\sum_{i=1}^{N} y_i$$

   where:

   - Y is the final prediction of the Random Forest regressor.
   - N is the number of trees in the forest.
   - y_i is the prediction made by the i-th tree.

Important Parameters and Considerations

- Number of Trees (N): The number of trees in the forest. A larger number of trees can improve the performance but also increases computational complexity.
- Maximum Depth: The maximum depth of each tree. Deeper trees can capture more complex patterns but also can lead to overfitting.
- Minimum Samples Split: The minimum number of samples required to split an internal node.
- Minimum Samples Leaf: The minimum number of samples required to be at a leaf node.

Advantages of Random Forest

- Robustness: Can handle outliers and nonlinear data well.
- Versatility: Performs well on both regression and classification tasks.
- Handles High Dimensionality: Can handle thousands of input variables without variable deletion.
- Provides Feature Importance: Can output the importance of each feature for the prediction.

Disadvantages of Random Forest

- Model Size: The resulting model can be quite large and requires significant memory/storage.

- Computationally Intensive: More trees lead to more computation during both training and prediction.
- Less Intuitive: The ensemble nature of the model makes it harder to interpret than a single decision tree.

Code Implementation

The implementation of the Random Forest Regressor is detailed below using Python’s scikit-learn library:

```
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
import pandas as pd


data = pd.read_excel('/content/Combined_Polymer_Dataset__murari_singh.xlsx'
    red↪ )

# Define the numerical and categorical features based on the dataset
numerical_cols = [
   'density␣of␣polymer(g/cm^3)',
   "Young's␣modulus␣of␣matrix(Mpa)",
   'Tensile␣strength␣of␣matrix(Mpa)',
   'CNT␣weight␣fraction(%)',
   'CNT␣density(g/cm^3)',
   'CNT␣average␣dia(nm)',
   'CNT␣average␣length(nm)',
   "Young's␣modulus␣of␣CNTs(Gpa)"
]

categorical_cols = [
   'Polymer_matrix',
   'Processing_method',
   'CNT_surface_modification_method'
]

# Create the preprocessing pipelines for both numerical and categorical
    red↪ data
numerical_transformer = StandardScaler()
categorical_transformer = OneHotEncoder()

# Bundle preprocessing for numerical and categorical data
preprocessor = ColumnTransformer(
   transformers=[
      ('num', numerical_transformer, numerical_cols),
      ('cat', categorical_transformer, categorical_cols)
   ])

# Split the data into features and target
X = data.drop('Tensile␣strength␣of␣the␣nano-composites(Mpa)', axis=1)
Y = data['Tensile␣strength␣of␣the␣nano-composites(Mpa)']

# Create a pipeline that combines the preprocessor with a
    red↪ RandomForestRegressor
model = Pipeline(steps=[('preprocessor', preprocessor),
                        ('regressor', RandomForestRegressor(random_state=42))])

# Split the data into training and testing sets
```

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,
    red↪ random_state=42)

# Fit the model pipeline with the training data
model.fit(X_train, Y_train)

# Now the model and its preprocessing steps are fitted and can be used to
    red↪ make predictions
# Predict using the fitted model pipeline
predicted_tensile_strength = model.predict(X_test)

# Print the predicted values
print(predicted_tensile_strength)
```

3 Model Evaluation

3.1 Performance On Training Set

The model's performance was evaluated on the training set using various metrics to understand its accuracy and reliability. The following Python code calculates the Mean Squared Error (MSE), Root Mean Squared Error (RMSE), R-squared (R²), and Standard Deviation of Residuals:

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

X = X_encoded
Y = data['Tensile␣strength␣of␣the␣nano-composites(Mpa)']

# Splitting the dataset into training and development sets
X_train, X_dev, Y_train, Y_dev = train_test_split(X, Y, test_size=0.2,
    red↪ random_state=42)

# Creating the Random Forest Regressor model
random_forest_model = RandomForestRegressor(random_state=42)

# Training the model on the training data
random_forest_model.fit(X_train, Y_train)

# Predicting on the training set to evaluate the performance
Y_train_pred = random_forest_model.predict(X_train)

# Calculating performance metrics on the training set
mse_train = mean_squared_error(Y_train, Y_train_pred)
rmse_train = np.sqrt(mse_train)
r2_train = r2_score(Y_train, Y_train_pred)
std_train = np.std(Y_train - Y_train_pred)

# Printing the performance metrics for the training set
print("Model␣Performance␣on␣the␣Training␣Set:")
print(f"Mean␣Squared␣Error␣(MSE):␣{mse_train:.4f}")
print(f"Root␣Mean␣Squared␣Error␣(RMSE):␣{rmse_train:.4f}")
print(f"R-squared␣(R):␣{r2_train:.4f}")
print(f"Standard␣Deviation␣of␣Residuals:␣{std_train:.4f}")
```

The model's performance on the training set is as follows:

- Mean Squared Error (MSE): 1.4926

- Root Mean Squared Error (RMSE): 1.2217
- R-squared (R²): 0.9994
- Standard Deviation of Residuals: 1.2173

3.2 Evaluated the model on the development set

Evaluated the model on the development set using metrics such as Mean Squared Error (MSE) and R-squared (R²).

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Creating the Random Forest Regressor model
random_forest_model = RandomForestRegressor(random_state=42)

# Training the model on the training data
random_forest_model.fit(X_train, Y_train)

# Predicting on the development set
Y_dev_pred = random_forest_model.predict(X_dev)

# Evaluating the model
mse = mean_squared_error(Y_dev, Y_dev_pred)
r2 = r2_score(Y_dev, Y_dev_pred)

print("Model␣Performance␣on␣the␣Development␣Set:")
print(f"Mean␣Squared␣Error:␣{mse}")
print(f"R-squared:␣{r2}")
```

The model's performance on the development set is as follows:
- Mean Squared Error: 23.654280061577015
- R-squared: 0.9923401407919821

3.3 Cross-Validation

Performed 5-fold cross-validation to assess the model's stability and reliability. The cross-validation scores provided insights into the model's consistency.

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import cross_val_score
import numpy as np

# Creating the Random Forest Regressor model
random_forest_model = RandomForestRegressor(random_state=42)

# Applying k-fold cross-validation (let's use k=5)
k = 5
cv_scores = cross_val_score(random_forest_model, X, Y, cv=k, scoring='
    red↪ neg_mean_squared_error')

# Converting the scores to positive values (as they are returned as
    red↪ negative by convention)
cv_scores = np.abs(cv_scores)

# Calculating mean and standard deviation of the scores
mean_cv_scores = np.mean(cv_scores)
std_cv_scores = np.std(cv_scores)

print(f"Cross-Validation␣Scores␣(MSE)␣for␣{k}␣folds:␣{cv_scores}")
print(f"Mean␣CV␣MSE:␣{mean_cv_scores}")
print(f"Standard␣Deviation␣of␣CV␣MSE:␣{std_cv_scores}")
```

The cross-validation results are as follows:

- Cross-Validation Scores (MSE) for 5 folds: [29.79966164 12.91104918 11.94297347 3.42427634 5.5295586]
- Mean CV MSE: 12.721503846493459
- Standard Deviation of CV MSE: 9.278663982158818

4 Result

4.1 Visualization of the Scatter Plot

Created scatter plots to compare actual vs. predicted values, providing a visual understanding of the model's predictive accuracy.

```
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Creating the Random Forest Regressor model
random_forest_model = RandomForestRegressor(random_state=42)

# Training the model on the training data
random_forest_model.fit(X_train, Y_train)

# Predicting on the development set
Y_dev_pred = random_forest_model.predict(X_dev)

# Evaluating the model
mse = mean_squared_error(Y_dev, Y_dev_pred)
r2 = r2_score(Y_dev, Y_dev_pred)

print("Model␣Performance␣on␣the␣Development␣Set:")
print(f"Mean␣Squared␣Error:␣{mse}")
print(f"R-squared:␣{r2}")

# Plotting the actual vs predicted values
plt.figure(figsize=(10, 6))
plt.scatter(Y_dev, Y_dev_pred, alpha=0.75)
plt.plot([Y_dev.min(), Y_dev.max()], [Y_dev.min(), Y_dev.max()], 'k--', lw
    red↪ =3)
plt.xlabel('Actual␣Tensile␣Strength␣(Mpa)')
plt.ylabel('Predicted␣Tensile␣Strength␣(Mpa)')
plt.title('Actual␣vs␣Predicted␣Tensile␣Strength␣of␣Nano-Composites')
plt.show()
```

Model Performance on the Development Set:

- Mean Squared Error: 23.654280061577015
- R-squared: 0.9923401407919821

Few predicted tensile strength (Mpa):

```
[111.50827577, 98.41517732, 108.94976732, 132.74882114, 98.79548225,
132.96175193, 102.79277645, 112.81493763, 120.81813024, 125.97003489,
124.66254784, 104.88757477, 130.14195713, 90.08963324, 120.38773188,
110.6398798, 116.6289562, 74.43085722, 93.11327113, 106.09361032]
```

4.2 User Interface For Prediction

A user interface was developed to input the features and predict the tensile strength of the nano-composite using the model. The code for this user interface is as follows:

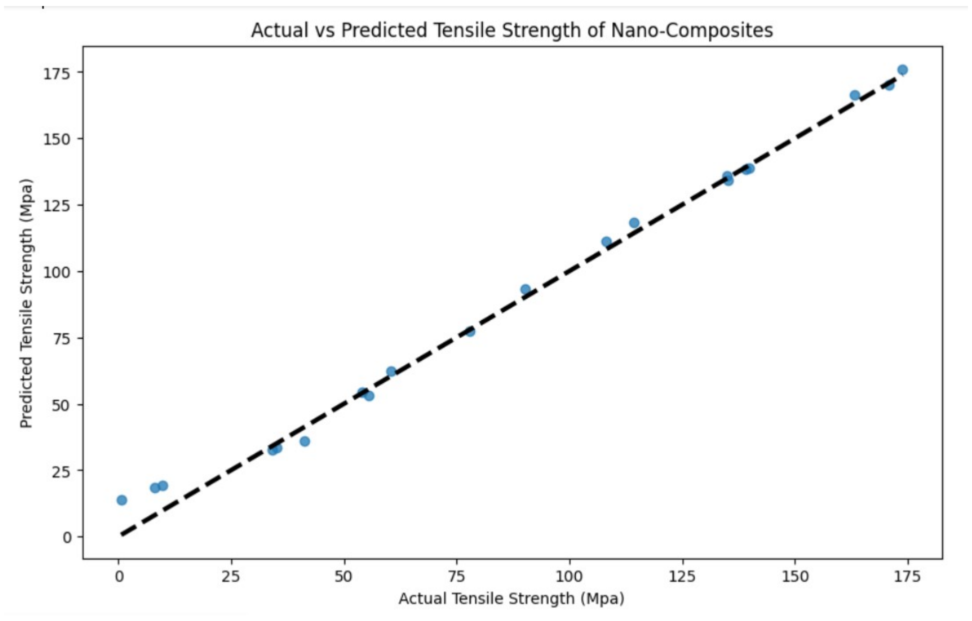


Figure 12: Scatter Plot of Actual vs Predicted Tensile Strength.

```
def prompt_and_predict(model):
    # Create a dictionary for the input features
    input_features = {}

    # Prompt for numerical features
    for feature in numerical_cols:
        input_features[feature] = float(input(f"Enter␣{feature}:␣"))

    # Prompt for categorical features
    for feature in categorical_cols:
        input_features[feature] = input(f"Enter␣{feature}:␣").strip()

    # Convert the dictionary to a DataFrame
    input_df = pd.DataFrame([input_features])

    # Make a prediction using the model
    prediction = model.predict(input_df)

    return prediction[0]

# Prompt the user for input and make a prediction
tensile_strength_prediction = prompt_and_predict(model)

# ANSI escape sequences to start and end the colored text (green text here)
start_green = "\033[92m"
end_green = "\033[0m"

print(f"The␣predicted␣tensile␣strength␣of␣the␣nano-composite␣is:␣{
    red↪ start_green}{tensile_strength_prediction}{end_green}␣Mpa")
```

Result Of Dry Run

The following interaction demonstrates a dry run of the user interface, providing an example of how predictions are generated:

```
Enter density of polymer(g/cm^3): .98
Enter Young’s modulus of matrix(Mpa): 205
Enter Tensile strength of matrix(Mpa): 101
Enter CNT weight fraction(%): 18
Enter CNT density(g/cm^3): 2.2
Enter CNT average dia(nm): 97
Enter CNT average length(nm): 10000
Enter Young’s modulus of CNTs(Gpa): 800
Enter Polymer_matrix: Epoxy
Enter Processing_method: Ball milling
Enter CNT_surface_modification_method: Amine-modified
The predicted tensile strength of the nano-composite is: 87.18052313788613 Mpa
```

4.3 Error Handling and Improvements

In the development of the user interface and the prediction model, certain error handling and improvements were implemented to enhance the robustness and reliability of the system.

- Modified the `OneHotEncoder` to handle unknown categories in the input data. This modification ensures that the model can process and make predictions even when it encounters unseen categorical data during prediction. The `OneHotEncoder` was set with the parameter `handle_unknown='ignore'` to achieve this.

  ```
  from sklearn.preprocessing import OneHotEncoder
  onehot_encoder = OneHotEncoder(handle_unknown='ignore')
  ```

- Ensured that input features are correctly processed before making predictions. This involves normalizing the numerical data and one-hot encoding the categorical data. Proper preprocessing of input features is crucial for the model to interpret the data correctly and make accurate predictions.

  ```
  # Normalizing numerical features and one-hot encoding categorical
     red↪ features
  input_df_normalized = scaler.transform(input_df[numerical_cols])
  input_df_encoded = onehot_encoder.transform(input_df[categorical_cols])
  ```

These improvements are aimed at making the model more flexible and capable of handling a variety of input data scenarios, thereby increasing its usability and effectiveness.

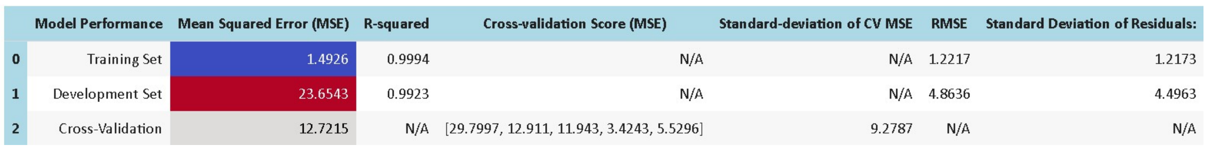
5 Conclusion

The Random Forest model was successfully developed and evaluated. It can predict the tensile strength of nano-composites based on various input features. The model shows good performance based on MSE and R² metrics and demonstrates consistency across different subsets of data as evidenced by cross-validation scores.

The interactive prediction function enhances the practical application of the model, allowing users to input new data and receive predictions in real time. The implementation of preprocessing steps ensures that the model handles new data correctly.

5.1 Model Performance Metrics

The following image presents the performance metrics of the developed model:



	Model Performance	Mean Squared Error (MSE)	R-squared	Cross-validation Score (MSE)	Standard-deviation of CV MSE	RMSE	Standard Deviation of Residuals:
0	Training Set	1.4926	0.9994	N/A	N/A	1.2217	1.2173
1	Development Set	23.6543	0.9923	N/A	N/A	4.8636	4.4963
2	Cross-Validation	12.7215	N/A	[29.7997, 12.911, 11.943, 3.4243, 5.5296]	9.2787	N/A	N/A

Figure 13: Model Performance Metrics

5.2 Comparison with Other Models

This subsection presents a visual comparison between the performance of the Random Forest algorithm and the Gaussian Process Regression model.

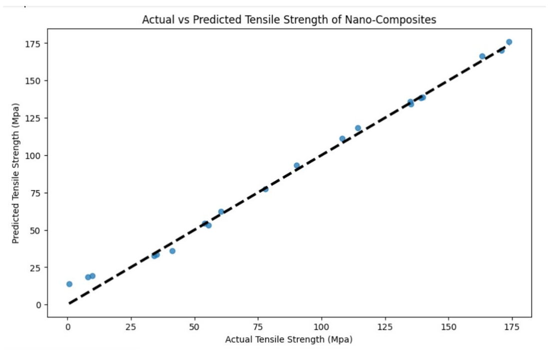


Figure 14: Random Forest Algorithm Results

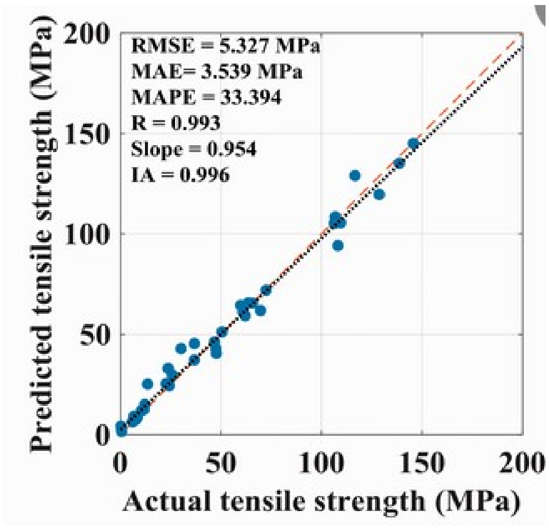


Figure 15: Gaussian Process Regression Results

6 Future Work

In the forthcoming semester, I will obtain a much larger dataset from DRDO. The plan is to implement a deep learning model that will allow us to predict outcomes with greater accuracy and handle a larger number of input features.

The following points outline the intended future work:

- Fine-tuning of model parameters to achieve optimal performance.
- Testing the model with a broader set of data to assess its generalizability and robustness.
- If necessary, implementing additional features or exploring more advanced machine learning algorithms to improve predictive capabilities.
- Developing a graphical user interface (GUI) to facilitate easier interaction with the model, especially for non-technical users, to make the application of the model more accessible and user-friendly.

This future work aims to enhance the model's performance and usability, thereby extending its application to a wider range of real-world problems and users.

7 References

References

[1] François Fleuret. *The Little Book of Deep Learning*, 2023.

[2] R. Ponnusamy, S. Sathyamoorthy, & K. Manikandan. *A Review of Image Classification Approaches and Techniques*, 2020.

[3] Manas Ranjan Swain and Ramesh Chandra Mohanty. *Prediction of Mechanical Properties of Polymer Composites using Machine Learning Techniques: A Review*. Materials Today: Proceedings, 2021.

[4] D. P. Kingma, & J. Ba. *Adam: A method for stochastic optimization*. arXiv preprint arXiv:1412.6980, 2014.

[5] B. Adhikari, T. J. Jadhao, & B. K. Das. *Machine learning applications in composite materials: A review*. Composites Part B: Engineering, 2020.

[6] J.N. Coleman, U. Khan, Y.K. Gun'ko. *Mechanical reinforcement of polymers using carbon nanotubes*.

[7] W. Bauhofer, J.Z. Kovacs. *A review and analysis of electrical percolation in carbon nanotube polymer composites*.

[8] Z.D. Han, A. Fina. *Thermal conductivity of carbon nanotubes and their polymer nanocomposites: a review*.

[9] A. Paipetis, V. Kostopoulos. *Carbon Nanotube Enhanced Aerospace Composite Materials: A New Generation of Multifunctional Hybrid Structural Composites*.

[10] D.D.L. Chung. *Carbon materials for structural self-sensing, electromagnetic shielding and thermal interfacing*.