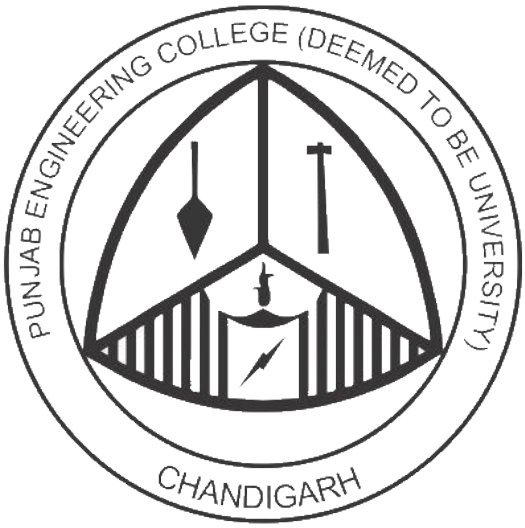

Machine Learning Project

Predicting Tensile Strength of Aluminium alloys using Regression techniques

By:

Rohan G Hugar

M.Tech. Mechanical Engineering

SID: 22209002

Abstract

Knowledge of the mechanical properties of engineering materials is essential for their practical applications. In the present project, one-hundred and seventy three data samples on three mechanical properties of different grades of Aluminium alloys —2% proof stress, tensile strength and % elongation —were selected from the Japan National Institute of Material Science database, comprising data on Aluminium alloys. Three machine learning algorithms were used to predict the tensile strength property of the materials represented by the one-hundred and seventy three data samples, and Bagging regressor showed the best predictive performance.

Index

Sr No.	Chapter/ Fig	Contents	Page No
1	0	Abstract	1
2	1	Introduction	4
3	1	Data Source	4
4	2	Problem Statement	5
5	Fig 1.	Stress-Strain curve for Al 2000 grades.	5
6	3	Dataset Details	6
7	Fig 2.	.head( ) function on pandas data	6
8	4	Preprocessing Techniques	7
9		Inter Quantile Range (IQR)	7
10	Fig 3.	Dataset Description	8
11		MinMaxScalar	8
12	Fig 4.	Final Dataset	8
13	5	Model Implementation	9
14		Random Forest	9
15		Bagging Regressor	9
16		Linear Regressor	10
17	6	Comparing all models	11
18	Fig 5.	Random Forest	11
19	Fig 6.	Bagging Regressor	11
20	Fig 7.	Linear Regressor	12
21	Fig 8.	MAE comparison	12
22	Fig 9.	R2 score comparison	12
23	7	Conclusion	13

Chapter 1: Introduction

The identification of structure-property relationships is fundamental to the discovery of new materials. However, the ability to comprehensively understand and manipulate structure-property relationships of materials is very challenging, due to the diversity and complexity of materials. As a result, data-driven discovery of novel advanced materials requires the use of advanced techniques such as artificial intelligence (AI) and machine learning (ML) to accelerate research and development. Materials data and ML provide the foundation of this data-driven materials discovery paradigm, which integrates materials domain knowledge and artificial intelligence technology to form the new research field of materials informatics.

The purpose of this project is to predict the mechanical properties of Al alloys using ### ML algorithms, especially using Random Forest (RF) regression. The performances of the ### algorithms were assessed, revealing that RF performing the best.

Data Source

The publicly available dataset for steels in the Japan National Institute of Material Science (NIMS) was used in this study, as it is among the world's largest experimental datasets of its type. The NIMS dataset contains materials chemical compositions, processing conditions and property information, including the mechanical properties of Al alloys (among many other engineering materials) at room temperature, such as fatigue strength, tensile strength, fracture strength, 2% proof stress, % elongation, hardness etc.

Chapter 2: Problem Statement

The purpose of this project is to predict the Tensile strength of Aluminium alloys. Tensile strength refers to the maximum amount of stress that a material can withstand before breaking or deforming under tension. In the context of aluminium, tensile strength is a measure of its ability to resist being pulled apart or stretched when a force is applied to it.

Aluminium has a high tensile strength, which means that it can withstand significant amounts of stress before breaking or deforming. The exact value of the tensile strength of aluminium depends on the specific alloy and temper of the material, as well as the manufacturing process used. However, typical tensile strengths for aluminium alloys used in structural applications range from around 270 MPa to 620 MPa (Mega Pascal).

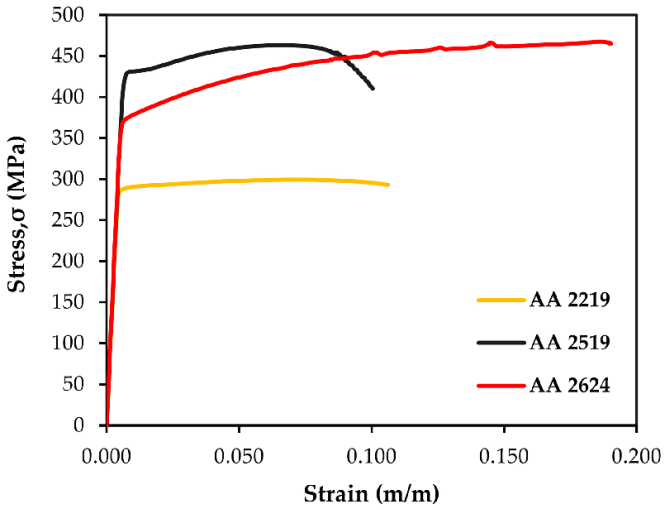

Fig 1: Stress-Strain curve for Al 2000 grades.

The dataset for the project is taken from publicly available repository from Japan National Institute of Material Sciences (NIMS).

Chapter 3: Dataset Details

The dataset is taken from the Japan National Institute of Material Science (NIMS) which is open-source and contains various datasets comprising of engineering metals' chemical, mechanical, thermal, vibrational, physical, magnetic and metallurgical properties. In this project, the mechanical properties of Aluminium alloys dataset having One Hundred Seventy Three datapoint and Sixteen attributes is used. This raw data is refined for our specific use case. The attributes of this dataset are:

```
['ID','X','Fe (wt%)','Mn (wt%)','Si (wt%)','Al (wt%)','Mg
(wt%)','Ti (wt%)','Cu (wt%)','Cr (wt%)','V (wt%)','Zr (wt%)','Zn
(wt%)','2% proof stress (Mpa)','Tensile strength (Mpa)',
'Elongation (%)']
```

[Element ID, identifier, Iron, Manganese, Silicon, Aluminium, Magnesium, Titanium, Copper, Chromium, Vanadium, Zirconium, Zinc, 2% Proof Stress, Tensile Stress (in MPa) and % elongation] respectively.

In materials science and engineering, the 2% proof stress (also known as yield strength) is the stress at which a material begins to deform plastically, i.e., it undergoes permanent deformation even after the stress is removed. For aluminium, the 2% proof stress is the stress at which the material starts to yield or deform plastically by 2% of its original length or cross-sectional area.

The 2% proof stress is an important mechanical property of aluminium, as it provides a measure of the material's ability to resist deformation under load. It is commonly used to specify the strength of aluminium alloys in engineering applications.

```
df_raw = pd.read_csv('al-alloys.csv')
df_raw.head()
```

	ID	X	Fe (wt%)	Mn (wt%)	Si (wt%)	Al (wt%)	Mg (wt%)	Ti (wt%)	Cu (wt%)	Cr (wt%)	V (wt%)	Zr (wt%)	Zn (wt%)	2% proof stress (Mpa)	Tensile strength (Mpa)	Elongation (%)
0	A 5005 P	1	0.35	0.1	0.15	98.33	0.8	0.0	0.1	0.05	0.0	0.0	0.125	95	125	2
1	A 5005 P	1	0.35	0.1	0.15	98.33	0.8	0.0	0.1	0.05	0.0	0.0	0.125	120	145	2
2	A 5005 P	1	0.35	0.1	0.15	98.33	0.8	0.0	0.1	0.05	0.0	0.0	0.125	145	165	2
3	A 5005 P	1	0.35	0.1	0.15	98.33	0.8	0.0	0.1	0.05	0.0	0.0	0.125	165	185	2
4	A 5005 P	2	0.35	0.1	0.15	98.33	0.8	0.0	0.1	0.05	0.0	0.0	0.125	85	120	4

Fig 2: Calling the .head( ) method in pandas to visualise dataset.

In order to predict the Tensile Strength of Aluminium alloys, the refining of the dataset has to be done by dropping the `'ID', 'X', '2% proof stress (Mpa)', Elongation (%)` columns. After reading the `.csv` file, the dataset is visualised by using the `.head()` method in pandas data-frame.

Chapter 4: Preprocessing Techniques

Data Preprocessing is one of the most important step in solving any Machine Learning problem. Preprocessing in machine learning refers to the steps taken to prepare data for use in a machine learning algorithm. Preprocessing can include a variety of tasks, such as data cleaning, normalisation, feature selection, feature engineering, and scaling.

Preprocessing is a crucial step in machine learning, as it can have a significant impact on the performance of the final model. By carefully preparing and cleaning the data, selecting relevant features, and engineering new ones, the resulting model can be more accurate, efficient, and effective.

For this strength prediction project, first the Exploratory Data Analysis (EDA) is done and gaining insights from the EDA, some data preprocessing steps are taken. The outliers are removed using the Inter-Quantile Range (IQR) outlier removal method.

Inter-Quantile Range (IQR)

IQR is one of the outlier removal method, which tends to remove outliers by the following methodology:

```
quantile1 --> 25 percentile
quantile2 --> 75 percentile
IQR --> quantile1 - quantile2 ==> 25 to 75 percentile
Now,
      outliers upper bound --> quantile1 - 1.5 * IQR
      outliers lower bound --> quantile2 + 1.5 * IQR
      remove all data-points external to the bounds.
```

```
df.describe()
```

	Fe (wt%)	Mn (wt%)	Si (wt%)	Al (wt%)	Mg (wt%)	Ti (wt%)	Cu (wt%)	Cr (wt%)	V (wt%)	Zr (wt%)	Zn (wt%)	Tensile strength (Mpa)
count	173.000000	173.000000	173.000000	173.000000	173.000000	173.000000	173.000000	173.000000	173.000000	173.000000	173.000000	173.000000
mean	0.227514	0.164393	0.253873	95.952659	2.153295	0.045318	0.186705	0.175723	0.001734	0.014682	0.817168	261.433526
std	0.068303	0.170069	0.199373	2.541454	1.249819	0.040870	0.417126	0.204431	0.009175	0.041993	1.887479	93.987149
min	0.060000	0.010000	0.050000	87.050000	0.050000	0.000000	0.020000	0.000000	0.000000	0.000000	0.015000	110.000000
25%	0.200000	0.050000	0.130000	95.630000	0.900000	0.000000	0.050000	0.050000	0.000000	0.000000	0.050000	200.000000
50%	0.200000	0.080000	0.200000	96.780000	2.250000	0.050000	0.050000	0.150000	0.000000	0.000000	0.100000	255.000000
75%	0.250000	0.250000	0.230000	97.680000	3.100000	0.080000	0.100000	0.250000	0.000000	0.000000	0.125000	305.000000
max	0.500000	0.750000	1.000000	99.190000	5.050000	0.170000	2.300000	1.150000	0.050000	0.150000	7.800000	580.000000

Fig 3: Dataset description.

The dataset description is as follows:

MinMax Scalar

MinMax Scalar is a data preprocessing technique used to scale numeric features between 0 and 1. The MinMaxScaler works by first identifying the minimum and maximum values of each feature in the dataset. It then scales each feature to a new range, where the minimum value is mapped to 0 and the maximum value is mapped to 1. All other values are scaled proportionally between these two endpoints.

The formula used for MinMaxScaler is:

```
X_scaled = (X - X_min) / (X_max - X_min)
```

```
final_df.describe()
```

	Fe (wt%)	Mn (wt%)	Si (wt%)	Al (wt%)	Mg (wt%)	Ti (wt%)	Cu (wt%)	Cr (wt%)	V (wt%)	Zr (wt%)	Zn (wt%)	Tensile strength (Mpa)
count	92.000000	92.000000	92.000000	92.000000	92.000000	92.000000	92.000000	92.000000	92.0	92.0	92.000000	92.000000
mean	0.659420	0.320899	0.588406	0.386838	0.553179	0.442391	0.410326	0.699565	0.0	0.0	0.586462	0.545491
std	0.256970	0.342161	0.307594	0.233133	0.223913	0.397861	0.280016	0.341605	0.0	0.0	0.292340	0.206823
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.0	0.000000	0.000000
25%	0.583333	0.068182	0.333333	0.279352	0.451613	0.000000	0.312500	0.320000	0.0	0.0	0.318182	0.444444
50%	0.583333	0.090909	0.800000	0.329960	0.580645	0.400000	0.375000	1.000000	0.0	0.0	0.545455	0.537037
75%	0.833333	0.659091	0.800000	0.512146	0.666667	0.800000	0.750000	1.000000	0.0	0.0	0.772727	0.685185
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	0.0	0.0	1.000000	1.000000

Fig 4: Final dataset after MinMaxScalar

Chapter 5: Model Implementation

Three models namely Random Forest Regressor, Bagging Regressor and Linear Regressor were used.

Model 1: Random Forest Regressor

The Random Forest Regressor is a machine learning algorithm that belongs to the family of ensemble methods. It is a supervised learning algorithm used for regression problems, where the goal is to predict a continuous target variable based on a set of input features.

The algorithm creates multiple decision trees and combines their predictions to make a final prediction. Each decision tree is created using a subset of the input features and a random sample of the training data. This helps to reduce overfitting and increase the generalisation of the model. During the training phase, the Random Forest Regressor algorithm creates a large number of decision trees, each using a different subset of features and training data. Each decision tree makes a prediction for the target variable, and the final prediction is calculated as the average of all the individual tree predictions.

The Random Forest Regressor algorithm has several advantages over other regression algorithms, such as:

1. It is robust to outliers and missing data.
2. It can handle a large number of input features and does not require feature scaling.
3. It can provide estimates of feature importance, which can be useful for feature selection.
4. It is less likely to overfit than individual decision trees.

Model 2: Bagging Regressor

Bagging Regressor is a machine learning algorithm that belongs to the family of ensemble methods. It is a supervised learning algorithm used for regression problems, where the goal is to predict a continuous target variable based on a set of input features.

The Bagging Regressor algorithm creates multiple models, each using a random subset of the training data. Each model is trained independently and produces a prediction for the target variable. The final prediction is calculated as the average of all the individual model predictions.

The Bagging Regressor algorithm has several advantages over other regression algorithms, such as:

1. It is robust to outliers and noise in the training data.
2. It can handle a large number of input features and does not require feature scaling.
3. It can provide estimates of feature importance, which can be useful for feature selection.
4. It can reduce overfitting and increase the generalisation of the model.

Model 3: Linear Regressor

Linear Regression is a supervised learning algorithm used for regression problems, where the goal is to predict a continuous target variable based on a set of input features. In Linear Regression, the relationship between the input features and the target variable is assumed to be linear, meaning that the target variable can be expressed as a linear combination of the input features.

A Linear Regressor model works by finding the best fit line that can represent the relationship between the input features and the target variable. The best fit line is determined by minimising the sum of the squared differences between the actual target values and the predicted values.

The equation for the best fit line in Linear Regression is represented as:

$$y = \beta 0 + (\beta 1 * x1) + (\beta 2 * x2) + \ldots + (\beta n * xn)$$

Chapter 6: Comparison of all Models

Model 1: Random Forest Regressor

```
mae_rf_train = mean_absolute_error(y_train, model_rf.predict(x_train))
mae_rf_test = mean_absolute_error(y_test, model_rf.predict(x_test))
mse_rf_train = mean_squared_error(y_train, model_rf.predict(x_train))
mse_rf_test = mean_squared_error(y_test, model_rf.predict(x_test))
lmse_rf_train = mean_squared_log_error(y_train, model_rf.predict(x_train))
lmse_rf_test = mean_squared_log_error(y_test, model_rf.predict(x_test))

print("Random Forest:")
print("Training Mean Absolute Error:", round(mae_rf_train, 4))
print("Test Mean Absolute Error:", round(mae_rf_test, 4))
print("Training Root Mean Squared Errror:", np.sqrt(mse_rf_train))
print("Testing Root Mean Squared Errror:", np.sqrt(mse_rf_test))
print("Training Mean Squared Log Errror:", round(lmse_rf_train,4))
print("Testing Mean Squared Log Errror:", round(lmse_rf_test,4))
print("Baseline Mean Absolute Error:", round(baseline_mae, 4))
```

```
Random Forest:
Training Mean Absolute Error: 0.0893
Test Mean Absolute Error: 0.1392
Training Root Mean Squared Errror: 0.10830064669797645
Testing Root Mean Squared Errror: 0.16630918389976498
Training Mean Squared Log Errror: 0.005
Testing Mean Squared Log Errror: 0.0127
Baseline Mean Absolute Error: 0.1596
```

Fig 5: Random Forest Errors

Model 2: Bagging Regressor

```
mae_br_train = mean_absolute_error(y_train, model_br.predict(x_train))
mae_br_test = mean_absolute_error(y_test, model_br.predict(x_test))

mse_br_train = mean_squared_error(y_train, model_br.predict(x_train))
mse_br_test = mean_squared_error(y_test, model_br.predict(x_test))

lmse_br_train = mean_squared_log_error(y_train, model_br.predict(x_train))
lmse_br_test = mean_squared_log_error(y_test, model_br.predict(x_test))


print("Bagging Regressor:")
print("Training Mean Absolute Error:", round(mae_br_train, 4))
print("Test Mean Absolute Error:", round(mae_br_test, 4))

print("Training Root Mean Squared Errror:", np.sqrt(mse_br_train))
print("Testing Root Mean Squared Errror:", np.sqrt(mse_br_test))

print("Training Mean Squared Log Errror:", round(lmse_br_train,4))
print("Testing Mean Squared Log Errror:", round(lmse_br_test,4))

print("Baseline Mean Absolute Error:", round(baseline_mae, 4))
```

```
Bagging Regressor:
Training Mean Absolute Error: 0.0893
Test Mean Absolute Error: 0.1394
Training Root Mean Squared Errror: 0.10859593849089692
Testing Root Mean Squared Errror: 0.16327267099342815
Training Mean Squared Log Errror: 0.005
Testing Mean Squared Log Errror: 0.0123
Baseline Mean Absolute Error: 0.1596
```

Fig 6: Bagging Regressor Errors

Model 3: Linear Regressor

```
mae_lr_train = mean_absolute_error(y_train, model_lr.predict(x_train))
mae_lr_test = mean_absolute_error(y_test, model_lr.predict(x_test))

mse_lr_train = mean_squared_error(y_train, model_lr.predict(x_train))
mse_lr_test = mean_squared_error(y_test, model_lr.predict(x_test))

lmse_lr_train = mean_squared_log_error(y_train, model_lr.predict(x_train))
lmse_lr_test = mean_squared_log_error(y_test, model_lr.predict(x_test))


print("Linear Regressor:")
print("Training Mean Absolute Error:", round(mae_lr_train, 4))
print("Test Mean Absolute Error:", round(mae_lr_test, 4))

print("Training Root Mean Squared Errror:", np.sqrt(mse_lr_train))
print("Testing Root Mean Squared Errror:", np.sqrt(mse_lr_test))

print("Training Mean Squared Log Errror:", round(lmse_lr_train,4))
print("Testing Mean Squared Log Errror:", round(lmse_lr_test,4))

print("Baseline Mean Absolute Error:", round(baseline_mae, 4))
```

```
Linear Regressor:
Training Mean Absolute Error: 0.092
Test Mean Absolute Error: 0.1329
Training Root Mean Squared Errror: 0.11121386326773937
Testing Root Mean Squared Errror: 0.15503144787204423
Training Mean Squared Log Errror: 0.0052
Testing Mean Squared Log Errror: 0.0111
Baseline Mean Absolute Error: 0.1596
```

Fig 7: Linear Regressor

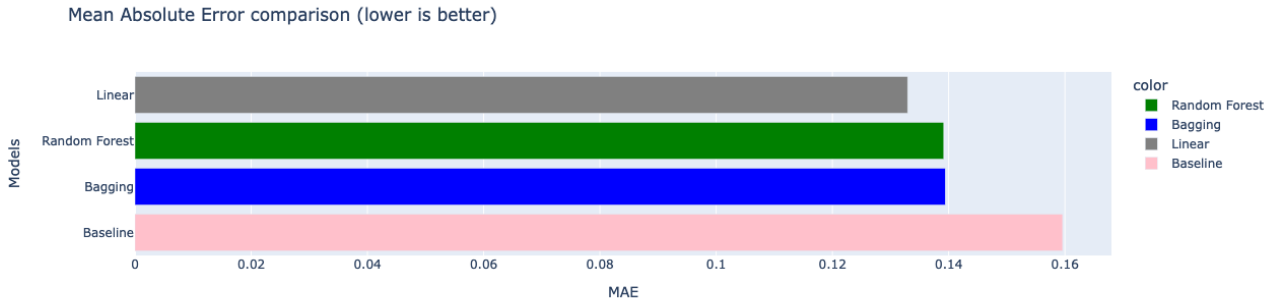

Fig 8. MAE comparison

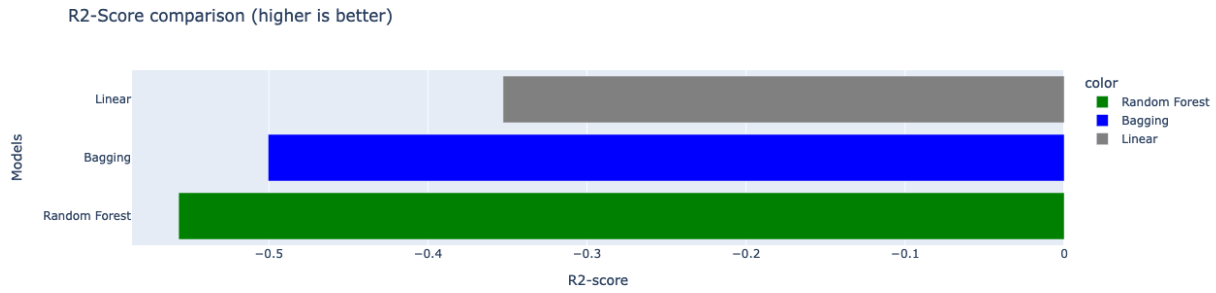

Fig 9. R2 score comparison

Chapter 7: Conclusion

Three machine learning algorithms were used to predict the tensile strength property of the materials represented by the one-hundred and seventy three data samples, and Bagging regressor showed the best predictive performance as the performance measure ie `r2 score` for Bagging Regressor is maximum and `mean absolute error` for Bagging regressor is the least. Although we can try out more Regressor techniques by simply using the `lazypredict library` from `sklearn.lazypredict` and importing `LazyRegressor()` further improving the metrics.