Building a fatigue strength predictor for steel using an ensemble deep learning model

A Project Report

submitted by

CHIRAG R BHATTAD

in partial fulfilment of the requirements
for the award of the degree of

MASTER OF TECHNOLOGY

DEPARTMENT OF METALLURGICAL AND MATERIALS ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

MAY 2019

THESIS CERTIFICATE

This is to certify that the thesis titled **Building a fatigue strength predictor for steel using an ensemble deep learning model**, submitted by **Chirag R Bhattad**, to the Indian Institute of Technology, Madras, for the award of the degree of **MASTERS OF TECHNOLOGY**, is a bona fide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

DDP Guide
Dr. Anand Krishna Kanjarla
Professor
Dept. of Metallurgical and Materials Engineering
IIT Madras, 600 036

Place: Chennai

Date: 6th May, 2019

ACKNOWLEDGEMENTS

I would like to take this opportunity to thank *Dr. Anand Kanjarla*, my DDP Guide, for assisting and guiding me throughout the project.

ABSTRACT

KEYWORDS: Fatigue Strength; Microstructure digitization; Material design; Material Informatics; Machine Learning; Regression; Artificial Neural Network; Deep Learning; Ensemble modelling.

Microstructure of a material is considered one of the best sources for generating data. It can be used to derive mechanical properties, physical properties as well as its chemical affinity. This makes tapping into the microstructure data an attractive prospect, with major advances in Material Informatics coming due to the use of this data. Thus, data-driven methods are emerging as an important toolset in the studies of multiscale, multiphysics, materials phenomena. Fatigue strength is one such property which can now be calculated without using destructive experimental methods. Here, I have described how an ensemble deep learning model reveals complex inter-relations between the features provided in the dataset and thus calculates the fatigue strength of steel using just five features after training on a dataset of 25 features. The choice of model was dictated by the small dataset of 437 data points extracted from the National Institute of Materials Science (NIMS), MatNavi, Japan. As many as 5 models have been chosen to form an ensemble deep learning model which predicts the fatigue strength of the dataset using just five feature as input. The model and the fatigue strength calculator has been hosted online for everyone to view and use at
https://fatigue-strength-model.herokuapp.com

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

IITM	Indian Institute of Technology, Madras
RTFM	Read the Fine Manual
NIMS	National Institute of Material Science
PSPP	Processing-Structure-Property-Performance
PSB	Persistent Slip Bands
PCA	Principal Component Analysis
SVM	Support Vector Machine
DoE	Design of Experiments
ODF	Orientation Distribution Function
RVE	Representative Volume Element
SVE	Statistical Volume Element
R-CNN	Region-Convolutional Neural Network
GAN	Generative Adversirial Network
MAE	Mean Absolute Error
RMSE	Root Mean Square Error
SDE	Standard Deviation Error
PIP	Python Installation Package
SVR	Support Vector Regression
CART	Classification And Regression Trees
MIC	Maximal Information Coefficient
RFE	Recursive Feature Elimination
PSB	Persistent Slip Bands

CHAPTER 1

INTRODUCTION

Material Science has traditionally been an experimental heavy field. It relied upon experimental observations and theoretical advancements to learn and expand the study of Material Science. But in the recent past, computational techniques and simulation-based advancements have become the primary source of new discoveries. They mimic the real experimental conditions using chunks of code on a digital screen. This code writes the rules of the experimental conditions and defines the relevant parameters to create a virtual experimental environment.

In the recent past, data generated from this virtual experimental simulations as well as real experiments have been funneled through a data pipeline to maintain a standard database which is ever-expanding. This has led to knowledge extraction via data-driven techniques, which is leading the modern revolution in Material Science. The unification of experimental, theoretical and computational Material science using data analytics and Machine learning techniques has heralded the emergence of a new field, Material Informatics.

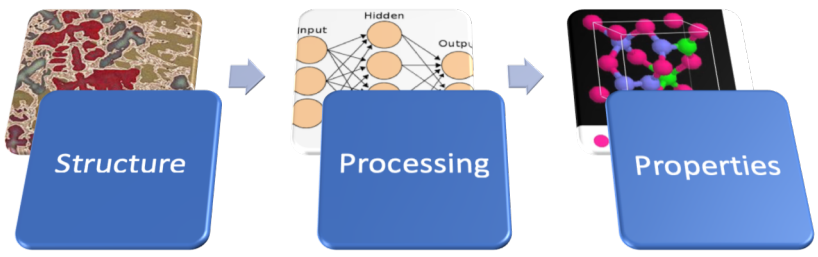

Figure 1.1: Digitization of microstructure is the goldmine leading to material innovation.

Material Informatics has been an ever expanding field of study and has been used for a wide array of applications, from predicting fatigue strength and Charpy toughness to discovering new and improved microstructures which can be mass produced and improved upon. The latter has been made possible by developing extremely sophisticated models which can unlock the processing-structure-property-performance (PSPP) using microstructural data.

This seemingly straight forward process has been hindered by the high dimensional representations needed for a rigorous description of the inherently heterogeneous material structure spanning multiple length or internal structure scales. Indeed, the precise physics-based connections between the material structure and its associated properties are very complex. However, from a practical viewpoint of materials design, it is imperative that we capture the high value information in these complex linkages in forms that allow computationally efficient explorations of the extremely large design spaces.

1.1 Fatigue Strength

Fatigue is the bane of every architect and civil engineer. It is defined as *"weakening of a material caused by the repeatedly applied loads."* Because of the repeated application of the load, the material fails, or experiences damage, at stress values much lower than the maximum stress it can withstand, known as the ultimate tensile stress limit. This phenomenon is due to the application of cyclic load, which causes localized structural damage to the material.

When the material is subjected to continuous loading, it can undergo a localized crack formation on the surface, grain interfaces, persistent slip bands (PSB) or interfaces of composites. Once the crack forms, the repeated cyclic load leads to crack propagation, which happens at a steady pace and goes unnoticed.

Once the crack reaches the critical limit, the crack will propagate at a rapid pace without any prior notice, causing sudden failure of the material. Thus, a new material property is defined for cyclic loading. This is known as fatigue strength. Fatigue strength is defined as the maximum stress that a material can withstand without breaking for a given number of cycles. Fatigue strength is an important structural property as it defines the stress amplitude that can cause material failure under cyclic conditions.

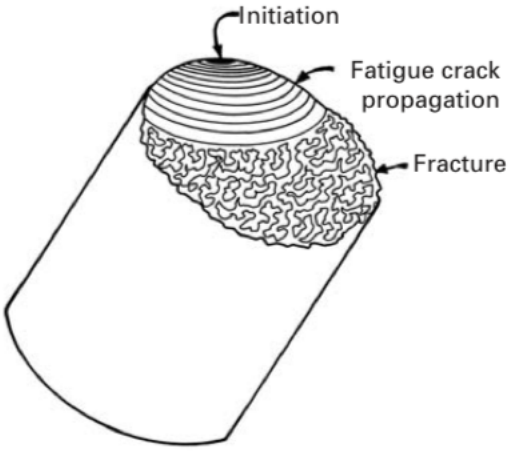

Figure 1.2: Initiation surface, propagation rate and fracture area of fatigue crack. Meyers and Chawla (2008)

The fracture occurring due to fatigue cycle is of brittle type, although it takes some time to propagate through the material. Theoretically, fatigue strength is a complex function of static stress values, chemical composition, microscopic structural data, environmental data, material processing parameters and loading cycle. Suresh (1998)

The current study has revealed new information with regards to the relationship between fatigue strength and material properties. The ensemble deep learning model revealed strong, data-driven, correlation between fatigue strength and material features such as Tempering temperature and chemical composition (% carbon, % phosphorous, % manganese and % chromium). This has led to the creation of a new and improved Fatigue Strength calculator which requires an input of just five features to calculate the fatigue strength of steel.

1.2 Data-driven approach

Data-driven approach for knowledge extraction has evolved leaps and bounds since the early 2000s. It combines the power of experimental, theoretical and simulational techniques to add a fourth dimension to material science innovation. In the current study, fatigue strength dataset from the National Institute of Material Science (NIMS), MatNavi, Japan is extracted to develop a fatigue calculator. In order to build this calculator, the data extracted from NIMS needs to be understood and visualized.

Sl. No.	NT	THT	THt	THQCr	CT	Ct	DT	Dt	QmT	TT	Tt	TCr	C
1	885	30	0	0	30	0	30	0	30	30	0	0	0.26
2	885	30	0	0	30	0	30	0	30	30	0	0	0.25
3	885	30	0	0	30	0	30	0	30	30	0	0	0.26
4	885	30	0	0	30	0	30	0	30	30	0	0	0.26
5	885	30	0	0	30	0	30	0	30	30	0	0	0.22
6	885	30	0	0	30	0	30	0	30	30	0	0	0.26
7	885	30	0	0	30	0	30	0	30	30	0	0	0.28
8	885	30	0	0	30	0	30	0	30	30	0	0	0.27
9	885	30	0	0	30	0	30	0	30	30	0	0	0.27
10	885	30	0	0	30	0	30	0	30	30	0	0	0.22
11	885	30	0	0	30	0	30	0	30	30	0	0	0.23
12	865	865	30	24	30	0	30	0	30	550	60	24	0.35
13	865	865	30	24	30	0	30	0	30	550	60	24	0.35
14	865	865	30	24	30	0	30	0	30	550	60	24	0.34
15	865	865	30	24	30	0	30	0	30	550	60	24	0.36
16	865	865	30	24	30	0	30	0	30	550	60	24	0.33
17	865	865	30	24	30	0	30	0	30	550	60	24	0.38
18	865	865	30	24	30	0	30	0	30	550	60	24	0.33
19	865	865	30	24	30	0	30	0	30	550	60	24	0.32
20	865	865	30	24	30	0	30	0	30	550	60	24	0.37
21	865	865	30	24	30	0	30	0	30	550	60	24	0.35
22	865	865	30	24	30	0	30	0	30	550	60	24	0.37
23	865	865	30	24	30	0	30	0	30	550	60	24	0.32

(a) Dataset - Part 1

Si	Mn	P	S	Ni	Cr	Cu	Mo	RedRatio	dA	dB	dC	Fatigue
0.21	0.44	0.017	0.022	0.01	0.02	0.01	0	825	0.07	0.02	0.04	232
0.18	0.44	0.009	0.017	0.08	0.12	0.08	0	610	0.11	0	0.04	235
0.27	0.43	0.008	0.015	0.02	0.03	0.01	0	1270	0.07	0.02	0	235
0.23	0.51	0.018	0.024	0.01	0.02	0.01	0	1740	0.06	0	0	241
0.19	0.42	0.026	0.022	0.01	0.02	0.02	0	825	0.04	0.02	0	225
0.22	0.45	0.026	0.026	0.02	0.05	0.03	0	825	0.06	0.02	0.02	245
0.27	0.5	0.021	0.018	0.02	0.03	0.01	0	1270	0.08	0.01	0	255
0.25	0.54	0.02	0.021	0.01	0.02	0.02	0	1740	0.07	0	0	245
0.23	0.37	0.008	0.03	0.06	0.1	0.09	0	610	0.08	0	0.04	275
0.26	0.47	0.021	0.023	0.02	0.02	0.01	0	1270	0.04	0.03	0	239
0.26	0.51	0.029	0.024	0.01	0.03	0.03	0	1740	0.11	0	0	255
0.21	0.77	0.021	0.022	0.01	0.01	0.02	0	825	0.05	0.03	0	397
0.3	0.74	0.017	0.024	0.01	0.1	0.01	0	1270	0.07	0.02	0	419
0.26	0.74	0.012	0.015	0.01	0.02	0.01	0	1740	0.08	0	0	401
0.26	0.7	0.009	0.023	0.08	0.12	0.09	0	610	0.07	0	0.03	394
0.21	0.75	0.031	0.019	0.01	0.03	0.02	0	825	0.08	0	0	426
0.29	0.7	0.01	0.03	0.06	0.09	0.11	0	610	0.06	0.01	0.01	426
0.26	0.76	0.023	0.013	0.01	0.1	0.03	0	1270	0.08	0.02	0	454
0.26	0.73	0.015	0.019	0.02	0.02	0.03	0	1740	0.06	0	0	418
0.27	0.78	0.016	0.02	0.01	0.11	0.02	0	1270	0.13	0.01	0	416
0.25	0.82	0.016	0.023	0.03	0.01	0.01	0	825	0.07	0.04	0	395
0.24	0.72	0.015	0.02	0.02	0.02	0.02	0	1740	0.1	0	0	433
0.21	0.63	0.007	0.017	0.06	0.1	0.08	0	610	0.06	0.05	0.03	379

(b) Dataset - Part 2

Figure 1.3: Snapshot of the dataset.

Before laying down the pipeline, certain terminologies need to be defined:

1.2.1 Features

Most datasets consist of a bunch of inputs which are independent individual variables. Features are these input variable. These features are used as inputs into Machine learning models to make predictions. These features can also be created from other features - a process known as feature engineering. Each column in a dataset is an individual feature. The dimension of the dataset is defined by the number of features (or columns) along with the number of data-points (or rows). Murphy (2012)

1.2.2 Target variable

Generally, the last column in the dataset is known as the target variable. It is the output variable or the final feature in the dataset. Target variable can take on multinomial values in case of a classification problem or a range of values in case of a regression problem. Murphy (2012)

1.2.3 Data-point

Each individual row in the dataset is defined as a data-point. Each row consists of feature variable and target variable and a combination of these two constitutes a data-point. The number of data-points along with the number of features defines the shape of the dataset. Murphy (2012)

Abbreviation	Details
NT	Normalizing Temperature
THT	Through Hardening Temperature
THt	Through Hardening time
THQCr	Cooling rate for Through Hardening
CT	Carburization Temperature
Ct	Carburization time
DT	Diffusion Temperature
Dt	Diffusion time
QmT	Quenching media Temperature (for carburization)
TT	Tempering Temperature
Tt	Temperating time
TCr	Cooling rate for Tempering
C	% Carbon
Si	% Silicon
Mn	% Manganese
P	% Phsophorous
S	% Sulphur
Ni	% Nickel
Cr	% Chromium
Cu	% Copper
Mo	% Molybdenum
RedRatio	Reduction Ratio (Ingot to Bar)
dA	Area Proportion of Inclusions Deformed by Plastic Work
dB	Area Proportion of Inclusions Occurring in Discontinuous Array
dC	Area Proportion of Isolated Inclusions
Fatigue	Rotating Bending Fatigue Strength (10^7 Cycles)

Table 1.1: Lists of features in the dataset.

The dataset given above consists of 25 features and 437 data-points. The features provided in the dataset define the physical process undertaken to create the steel, its chemical composition and the environmental and experimental conditions in which the steel was formed. The last feature in the table is the target variable or the output variable.

It is the fatigue strength for steel measured for Rotating bending at 10^7 cycles. Clearly defined are the various temperature dependent physical steel-making processes like Tempering, Normalization, Carburization, Quenching and Through Hardening while diffusion is also facilitated. Meanwhile, the chemical composition of steel is tabulated with high precision and, as we will find out later, this is extremely important in creating the new and improved fatigue strength calculator.

The data-processing pipeline includes data normalization, data visualization using scatter-plots to see the distribution of a particular feature with respect to another feature. Correlation heatmap among the different features also helps in identifying closely related features/columns. Idreos *et al.* (2015)

An important data visualization tool is Principle component analysis (PCA). It reduces the dimensionality of the features, bringing the dataset down to 2-3 independent features which captures the variance of the original dataset with great accuracy. Abdi and Williams (2010). After visualizing the dataset using PCA, the given dataset is run through the following feature ranking algorithms to get a cursory gist of the feature importance before applying machine learning models:

1. Univariate Selection Ho (2006)
2. Recursive feature selection Chen and Jeong

The final step is splitting the dataset into training dataset and test dataset. This is done to help measure the accuracy of the machine learning models employed later on. The model tries to learn from the training dataset and apply what it has learned on the test dataset. The splitting of the dataset is done randomly, marking 80% of the dataset as training data with the remaining 20% becoming test data.

1.3 Data Modelling

Now that we have the dataset split into training data and test data, we start applying regression based machine learning models onto the training data to learn from it. After fitting the train data, the model is run on test data to understand the accuracy of the model. Before outlining the steps, the following terminology needs to be defined:

1.3.1 Machine Learning

Machine Learning is the science of making the computers learn and extrapolate without being explicitly programmed to give out certain results. Like the name suggests, it gives *machines* the ability to mimic a particular human trait - *the ability to learn*. Goodfellow *et al.* (2016)

1.3.2 Artificial neural network

Artificial neural network is aptly defined by the inventor of the first neuro-computer, Dr. Robert Hecht-Nielsen as:
"...a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs."
Simply put, Artificial neural network try and mimic the human brain by creating a layer of neurons between the input and output layers. These neurons perform mathematical manipulations on the input data to fit it with the output data. Goodfellow *et al.* (2016)

1.3.3 Deep learning

Deep learning is an extension of the artificial neural network and a sub-field of machine learning. Deep learning models have more than one intermediate layer of interconnected neurons between the input and output data. Thus, it is a complex derivation from the simplified artificial neural network. Deep Learning models are able to fit complex, seemingly unrelated features using a black-box type algorithm mimicking the human brain. Goodfellow *et al.* (2016)

1.3.4 Ensemble

Ensemble modeling is a machine learning technique of running two or more related yet different models on the same dataset. The results of these models are then filtered into a single value using voting, averaging or other statistical techniques. Ensemble modeling greatly improves the accuracy by increasing the collective accuracy of individual, unique models. It also reduces the bias and variance of different model to give normalized results. Dietterich (2000)

S. No.	Machine Learning model
1	Support Vector Machine (SVM) Regressor
2	Linear Regression
3	Polynomial Regression
4	Decision Tree Regressor
5	Neural Network
6	Random Forest Regressor
7	Gradient Boosting Regressor
8	ADA Boosting Regressor
9	LightGBM Regressor
10	XGBoost Regressor

Table 1.2: List of Machine Learning models implemented.

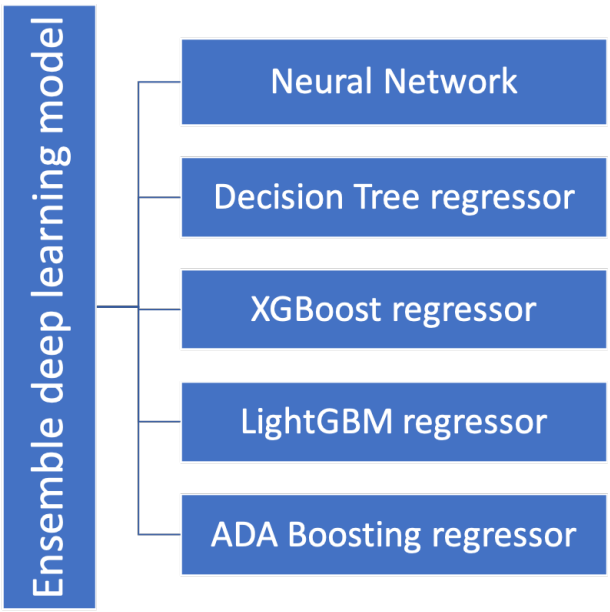

Figure 1.4: Constituents of the ensemble deep learning model.

There are a lot of machine learning models available to carry out regression algo-

rithms. For the current problem, we had a very small dataset with a reasonably small number of features. Thus, it is necessary to use algorithms which can identify the complex relationships between various features. Keeping this in mind, the following Machine Learning models were used in the current study to come up with a fatigue strength calculator:

After analyzing the results from the above models, a new ensemble deep learning model was created. This ensemble model chose only those machine learning models which were able to capture the variance of the original dataset. It also chose models which, in some sense, created complex feature inter-relations which were unique and non-overlapping.

CHAPTER 2

LITERATURE REVIEW

The traditional techniques used to solve the, still worked on, microstructure optimization problem have high time complexity and do not yield an exhaustive set of results. Traditional methods like Ashby plots like density vs strength plots have been used to discover new materials for a traditional application. But these methods have a narrow-minded approach with better models required to increase the result space.

Advanced data oriented techniques to enhance the mathematical search by statistical heuristics can be theoretically used to predict new materials with the same properties. Orientation Distribution Function (ODF) can be used to quantify the crystallographic texture. This is followed by a series of property optimization processes which develop heuristics that guide and nudge the search into the more promising areas. The heuristics are:

- BCC crystal plasticity model
- Random data generation.
- Search path refinement via feature selection.
- Search space reduction via classification schemes.
- Enhanced optimization

The efficiency of mathematical search deteriorates quickly as the candidate space grows, with a microstructure represented by hundreds, even thousands of dimensions making the traditional searches for microstructure design very slow. The main challenges are:

- **High dimensionality**: To search in a hypothetically unlimited space of all possible crystal orientation distributions, and converge within a reasonable time.
- **Multi-objective**: To optimize under a requirement of multiple external properties that are often in conflict.
- **Solution completeness**: To identify the complete space (or as much of it as possible) of microstructures when more than one solution exists that produce the same optimal property. Liu *et al.* (2015)

Property	eSearch	gSearch	ML	Percentage of time reduced
E	20054	4718.5	710.4	84.94%
Y	20145	7150.5	854.6	88.05%
m_s	21302	2588.0	422.5	83.67%
F_1	23225	1179.1	322.8	72.62%
F_2	25431	2783.3	589.4	78.82%

Figure 2.1: Comparing traditional techniques with heuristics based technique. Liu *et al.* (2015)

2.1 Material design

A comprehensive data-driven framework which can be used to perform analysis of materials has been developed. The framework is optimized for time and contains and exhaustive list of results for every iteration. This is achieved by integrating the three most important steps in microstructure optimization and material design. The three steps are:

- Design of experiments, where the input variables describing material geometry, phase properties and external conditions are sampled
- Efficient computational analyses of each design sample, leading to the creation of a material response database
- Machine learning applied to this database to obtain a new design or response model.

This integration leads to an optimized database, compact computation code and a clear choice of machine learning model to design and innovate new materials. The representation of microstructure data, it's storage, pre-processing and application of Kriging statistical model or Neural Network are all outlined for two different cases.

Traditional methods like finite element analysis of representative volume elements involves plasticity and damage. The use of "self consistent clustering analysis" which has been extensively used to build larger databases helps train the machine learning models better.

The use of *non-uniform space filling* design such as the *Sobol sequence* is an effective Experimental simulation. One key relation obtained sheds some light on the importance of the Design of Experiment (DoE). It is the relationship between the choice of

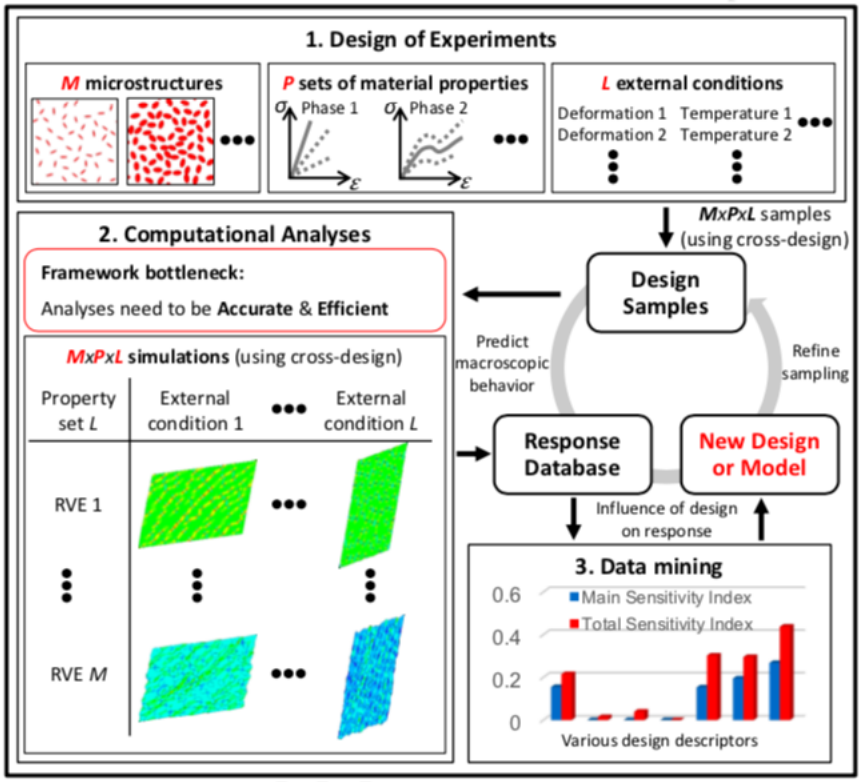

Figure 2.2: Schematic of global framework for data-driven material systems design/-modeling. Bessa *et al.* (2017)

DoE and the number of data points required to find the new design/model. Optimization of DoE can reduce the number of data-points, thus relieving some stress from the dynamic database by optimizing it and making it robust.

DoE is used as a starting point to propose the size and shape of the database. Once the database is established using Representative Volume Element (RVE) or Statistical Volume Element (SVE), it is run through one of the two machine learning models, *Kriging* and *Neural Networks*. The choice of the model is left to the application of the framework. Thus, a comprehensive set of rules are defined to develop a computational framework which will help accelerate material innovation and design. Bessa *et al.* (2017)

2.2 Machine Learning in Material Science

Machine Learning is becoming more and more embedded with the field of Material Science and Material design. Microstructure of a material can be converted into numbers to create an excel file which maps with an individual microstructure. This is followed

by series of standard steps taken to understand and regularize the dataset. It focuses on Dimensionality reduction methods like *Latent Variable reduction, Non-negative matrix factorization, Metric multi-dimensional data scaling* and *Feature extraction* to create a ready-to-use, non-biased, regularized dataset.

Emphasis is also given to data visualization and data exploration to identify edges and closed boundaries, cross-correlation and wavelets. These data-points help identify and mark outliers which can lead to erratic results by the machine learning models. A Clustering algorithm is also applied to mimic multi-phase microstructure to closest accuracy.

The potential application of machine learning models are:

- High-throughput phase diagrams
- Crystal structure determination
- Accelerated prediction of material property (like fatigue strength)
- Development and estimation of inter-atomic potential models
- Low-cost methods for material process control.

With quantum mechanical computations, development of density functions and Processing-Structure-Property-Performance (PSPP) modeling is also made possible by advanced computational models like the newly developed complex Neural Network models - Region-Convolutional Neural Network (R-CNN) and Generative Adversarial Network (GAN). Mueller *et al.* (2016)

2.3 Fatigue Strength calculator

National Institute of Materials Science (NIMS), MatNavi, Japan is one of the best sources for material science and metallurgical datasets. It also provides a 437 data-points dataset for fatigue strength which contains 25 feature columns.

This dataset is processed through a pipeline to normalize and scale it. The process includes data cleaning to remove null data-points, data scaling to bring all features into a uniform range and skewness removal, to ensure a Gaussian curve for all the features.

This data is then run through a series of regression based models to fit for Fatigue Strength. The effectiveness of these models is gauged by dividing the dataset into training data and test data and calculating the accuracy of the model by fitting it on train data and testing it out on test data. Some of the regression based techniques implemented are:

- Linear regression
- Multivariate Polynomial regression
- Decision Tree regression
- Boosting techniques like Gradient, ADA, Light and XGBoost
- bagging techniques like Artificial Neural Network and Random Forest
- Stacking and Ensemble techniques using the above listed models as base models

Table 2 Results comparison

Method	*R*	R^2	*MAE*	*RMSE*	*SDE*	MAE_f	$RMSE_f$	SDE_f
DecisionTable	0.9494	0.9014	34.8762	58.5932	47.1371	0.0584	0.0806	0.0557
IBk	0.9589	0.9195	46.0320	53.2749	26.8499	0.0859	0.0940	0.0382
KStar	0.9702	0.9413	36.9986	45.3779	26.3029	0.0706	0.0857	0.0487
SVM	0.9795	0.9594	24.2820	37.6250	28.7736	0.0400	0.0530	0.0349
LRTrans	0.9796	0.9596	22.3336	37.4748	30.1272	0.0370	0.0514	0.0357
RobustFitLSR	0.9804	0.9612	22.2152	37.2188	29.8960	0.0369	0.0520	0.0366
LinearRegression	0.9815	0.9633	25.6006	35.7168	24.9345	0.0456	0.0581	0.0360
PaceRegression	0.9816	0.9635	25.0302	35.5733	25.3065	0.0439	0.0565	0.0356
ANN	0.9861	0.9724	19.7778	31.0545	23.9695	0.0343	0.0470	0.0322
REPTree	0.9862	0.9726	22.5671	30.9401	21.1907	0.0414	0.0542	0.0349
M5ModelTree	0.9890	0.9781	19.3760	27.6065	19.6870	0.0353	0.0484	0.0332
MPR	0.9900	0.9801	18.5529	26.4378	18.8563	0.0350	0.0556	0.0432

Figure 2.3: Results of the top 12 models. Agrawal *et al.* (2014*a*)

The accuracy of the model is measured using r and r^2 score, Mean Absolute Error (**MAE**) - difference between two continuous variables, Root Mean Square Error (**RMSE**) - quadratic mean of the difference between two continuous variables and Standard Deviation Error (**SDE**) - Difference in Standard Deviation of two continuous variables. Agrawal *et al.* (2014*a*)

CHAPTER 3

MACHINE LEARNING MODELS

Arthur Samuel, who coined the term Machine Learning in the year 1959, defined it as "*the ability to learn without being explicitly programmed*". An elementary definition of Machine learning would describe it as:

1. Parsing data from the database
2. Using an algorithm to learn from it
3. Using this learnings to make a prediction. Goodfellow *et al.* (2016)

In it's most basic form, Machine learning is really just making a curve that best fits the data-points in a multi-dimensional representation of the data. Machine learning has taken every scientific field of study by storm, reducing the dependency on experimental results by creating a virtual experimental environment seamlessly and increasing the forward thrust by using the data generated by these models to create new theories and generating new ideas.

Machine learning models are classified into three broad categories based on the type of dataset it is working on. If the dataset consists of features **and** a target variable, then the model running on this dataset are called "Supervised learning" model. If the dataset only consists of features **without** any target variable, then the model running on this dataset are called "Unsupervised learning" model. If there is **no** dataset provided, then an action and reward/punishment environment is created and the model that runs in this environment is called a "Reinforcement learning" model. Kaelbling *et al.* (1996)

In the current study, 10 machine learning models were used to fit the training dataset. The models were chosen based on their complexity and their ability to preserve the variance of the dataset. From Linear regression to Neural Networks, 10 models were chosen to fit the data. Before taking a look at these models, a few words on the coding environment.

The current study used Jupyter notebook to write kernels of code, which acts as the *Integrated Development Environment*. The coding was done in *python programming*

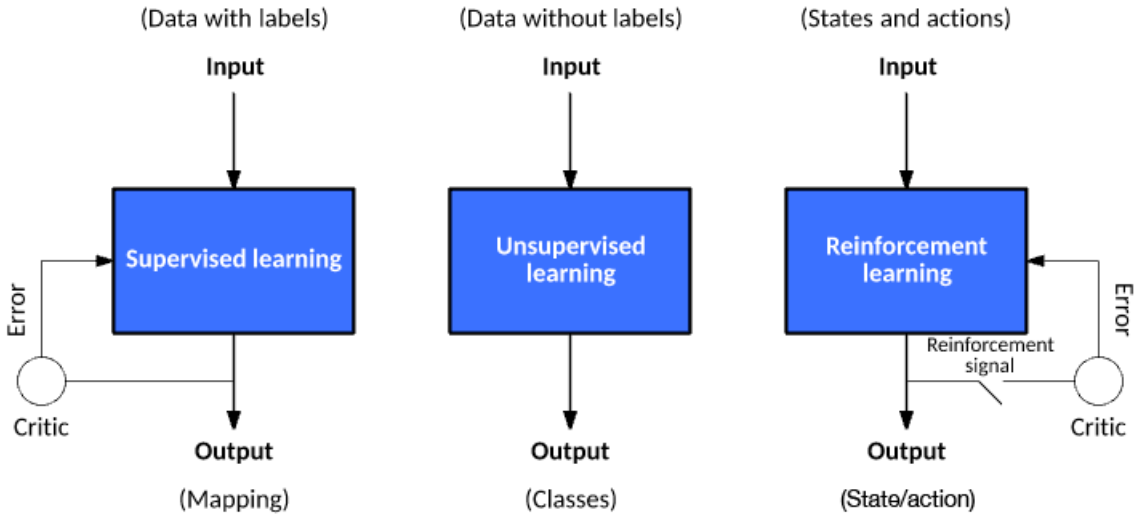

Figure 3.1: Different type of machine learning models.

language. So before diving into the different models used, we need to install certain python packages to run these models. To install these packages, an installer, specifically the **python installation package (pip)** needs to be installed to begin:

P.S: It is assumed that python programming language and Jupyter notebook is already installed in your computer.

```
sudo apt-get install python-pip python-dev build-essential
sudo pip install –upgrade pip
```

After installing **pip**, the libraries specified in the **Table 3.1** need to be installed. After installing them, the python libraries need to be imported to begin coding. Let us take a look at these models one by one.

Python Packages to be installed	Reason	code
pandas	Mathematical package	*pip install pandas*
numpy	Mathematical package	*pip install numpy*
matplotlib	Visualization package	*pip install matplotlib*
seaborn	Visualization package	*pip install seaborn*
pickle	Serializing package	*pip install pickle*
sklearn	Machine Learning package	*pip install -U scikit-learn*
tensorflow	Neural Networks	*pip install tensorflow*
keras	Neural Network	*pip install keras*
xgboost	Machine learning model	*pip install xgboost*
lightgbm	Machine learning model	*pip install lightgbm*

Table 3.1: List of python packages required to kick-start the study.

3.1 SVM Regressor

Support Vector Machines (SVM) are frequently used for classification problems, but they can also be used, to good effect, for regression problems. SVM becomes Support Vector Regression (SVR). Thus, Support Vector Machine can also be used as a regression method, maintaining all the main features that characterize the maximal margin algorithm.

The Support Vector Regression (SVR) uses the same principles as the SVM for classification, with only a few minor differences. Thus, it is important to understand how the SVM works in order to grasp the theory behind SVR.

"*A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data, the algorithm outputs an optimal hyperplane which categorizes new examples. In two dimensional space, this hyperplane is a line dividing the plane into two parts, where-in each class lay in either side.*"

This complex definition can be understood using this simple illustration:

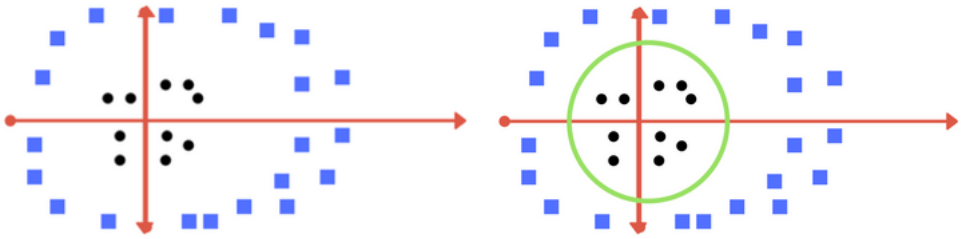

Figure 3.2: Illustration of Support Vector Machine.

Hyperplane is the line that will help us predict the continuous value or target value. There are two lines other than Hyperplane which creates a margin. These are the boundary lines. Support vectors can be on the Boundary lines or outside it. Keeping this definition in mind, here are the key points of difference between SVM and SVR:

1. Because the output is a real number and not a classification, it becomes very difficult to predict the information at hand, which has infinite possibilities.
2. In the case of regression, a margin of tolerance (epsilon) is set in approximation to the SVM which would have already requested from the problem.

The main idea is the same, to minimize error, individualizing the hyperplane which

maximizes the margin, keeping in mind that part of the error is tolerated. Ma and Guo (2014)

3.2 Linear Regression

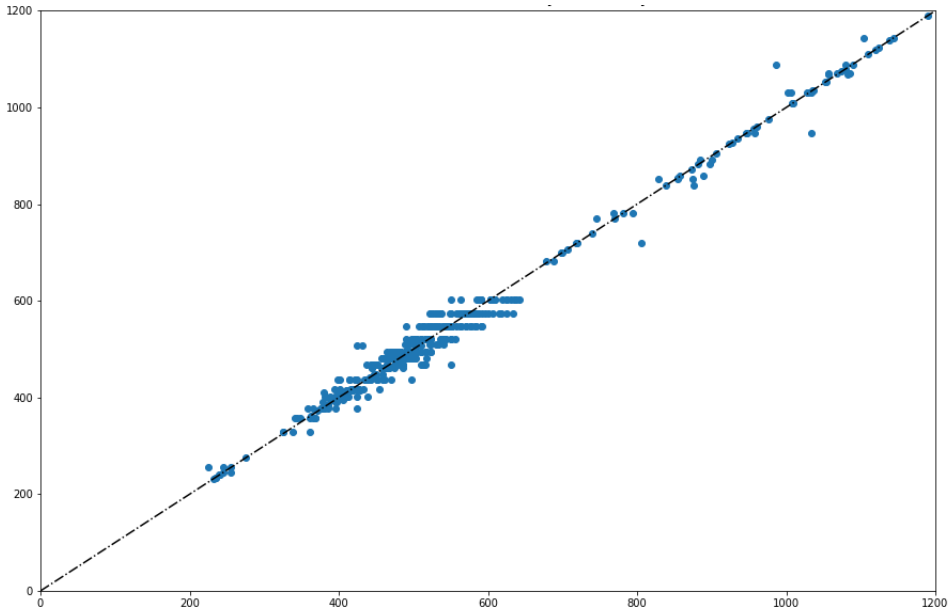

Figure 3.3: Illustration of Linear regression.

Linear regression is a linear mathematical equation used to find the relation between the independent variable/s (features) and dependent variable (target). It determines a statistical relationship between the two, which is not deterministic. At it's heart, linear regression tries to find the best line/plane that fits the given data by minimizing the error.

Error is defined as the distance between the predicted value on the regression line and actual value. The mathematical expression to define linear regression is:

$$y_i = \beta_0 1 + \beta_1 x_{i1} + ... + \beta_p x_{ip} + \epsilon_i = x_i^T \beta + \epsilon_i$$

The values of β must be chosen so as to minimize the error of the model. This is done by taking the sum of squares as the error function, which has to be reduced to optimize the model. Reduction in the error value leads immediately to a better fit of the model.

$$Error = \sum_{i=1}^{n} (actual_output - predicted_value)^2$$

Thus, linear regression, which uses a simplistic curve to fit the data, is a good regression model to fit the current dataset, but it is not the best due to the simple nature

and failure to uncover the complex correlation between features. Yan and Su (2009)

3.3 Polynomial Regression

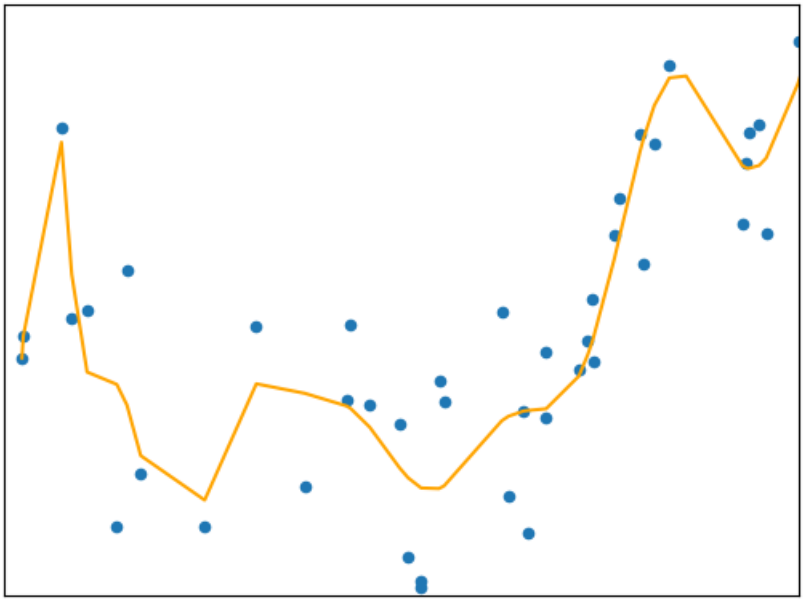

Figure 3.4: Illustration of Polynomial regression.

Polynomial Regression is a form of linear regression in which the relationship between the independent variable x and dependent variable y is modeled as an nth degree polynomial. Polynomial regression fits a nonlinear relationship between the value of x and the corresponding conditional mean of y, denoted E(y|x).

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + ... + \beta_n x^n + \epsilon$$

Although polynomial regression fits a nonlinear model to the data, as a statistical estimation problem it is linear, in the sense that the regression function E(y|x) is linear in the unknown parameters that are estimated from the data. For this reason, polynomial regression is considered to be a special case of multiple linear regression.

Polynomial Regression are basically used to define or describe non-linear phenomenon such as growth rate of tissues, progression of disease epidemics and distribution of carbon isotopes in lake sediments. Polynomial regression is used to avoid under-fitting and increase the complexity of the model in the cases where simple linear regression models are ineffective.

While polynomial regression reduces under-fitting, it runs the risk of over-fitting

the data, with the presence of even a few outliers causing major problems to the model. Thus, polynomial regression is very sensitive to outliers. This issue is deeply affected by the lack of algorithms to successfully detect and stamp out outliers from the dataset, making polynomial regression rather erratic and unpredictable. Cheng *et al.* (2018)

3.4 Decision Tree Regressor

As the name suggests, Decision tree regressor uses a tree like cascading structure to make decisions as you go down the tree. A Decision tree regressor breaks down the dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes.

The root node of each decision tree outlays the decision being made by that tree and it is called decision nodes. From the decision nodes, two or more branches appear, with each branch mirroring the values or range of values for a particular attribute.

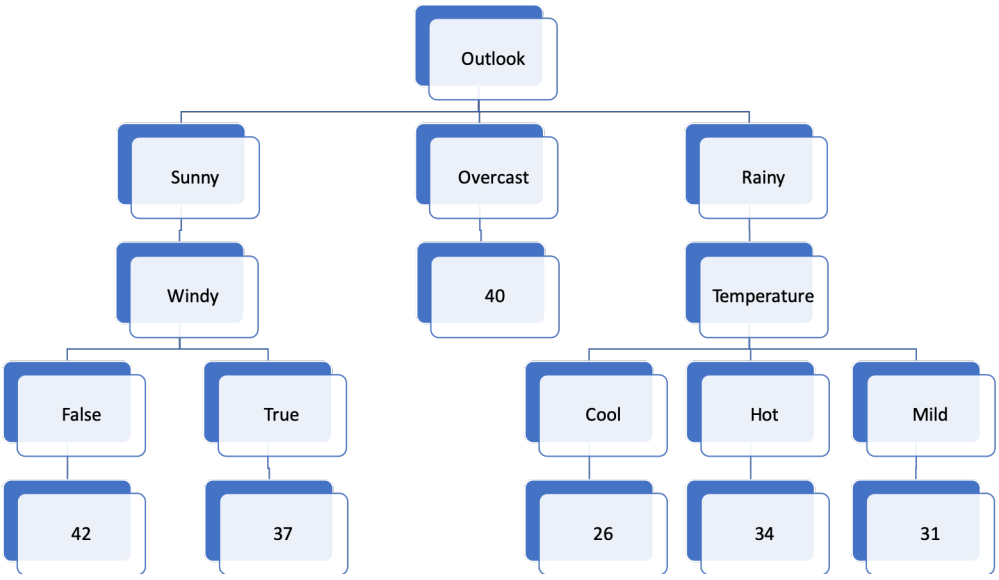

Figure 3.5: Example of a Decision tree.

The leaf node or the outermost nodes of the trees consist of values or categories assigned as you cascade through on of the branches of the decision tree. Decision tree breaks down the dataset into smaller and smaller subsets and tries to fit decision trees to these subsets. This makes the model extremely robust and generic. A comprehensive definition of decision trees is as follows:

"Decision Trees are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features."

While the above description points towards a clear application for classification problems, decision trees are extremely efficient and robust while being implemented on regression problems. The inner nodes develop range of values as value check for features and leads to a huge number of leaf nodes due to increased sophistication.

One important point which deters the use of decision tree is how unstable it can become even for small changes in the database. Given how it uses the whole range or classes of an attribute, any change in that attribute can destabilize the decision tree.

Decision trees are transparent in nature, with the whole tree accessible to the developer. This makes it easy to understand, which is why it is classified as a *"white box model"*. The decision-making is also precise with specific values/categories assigned at the end of a branch on the leaf node. Ultimately, decision tree regressor acts as one of the better models to work with in the current study. Quinlan (1986)

3.5 Neural Network

A neural network is a Machine Learning model made to mimic the human brain. The human brain has millions and millions of brain cells, called neurons. An Artificial neural network is a low level replication of this. An artificial neural network is a complex web of interconnected nodes which act like neurons. A neural network has the following basic parts:

1. Input layer
2. Hidden layer (activation function)
3. Output layer

Before explaining the significance of layers, the first step is understanding what a neuron is. Neurons are the basic computational units of a neural network. They take an input, perform some calculations and produce an output. A neuron has two essential components, weights and bias.

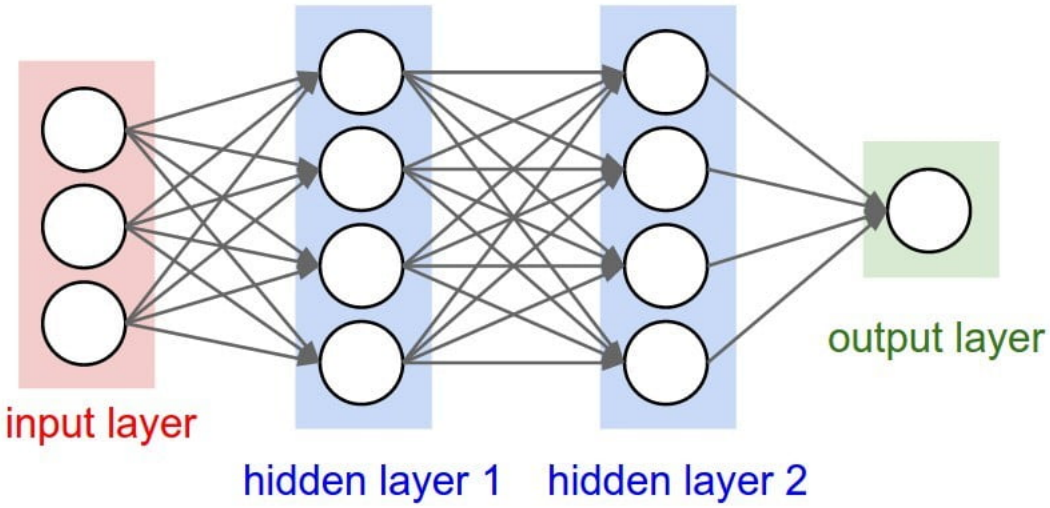

Figure 3.6: Representation of a simple Artificial Neural Network. Dormehl (2019)

Each layer is made of a particular number of neurons. Training of a neural network is nothing but the constant updation of the weights and bias associated with every neuron til we they achieve an optimal value which helps fit the dataset. This updation process is done via **forward propagation** of dataset through the model and **backward propagation** of error associated with the predictions through the model.

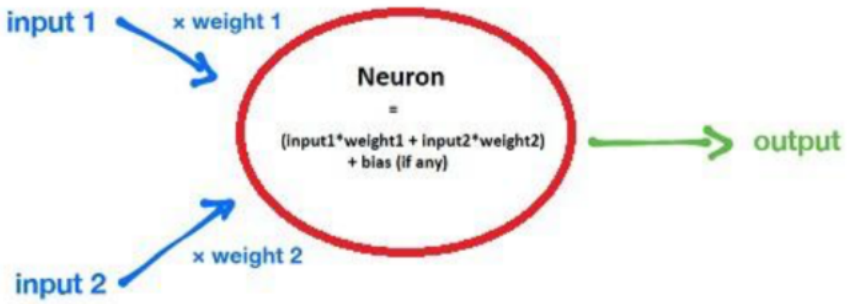

Figure 3.7: Representation of a Neuron.

The input layer is a unique layer containing a fixed number of neurons. This number is fixed because the number of neurons in an input layer is equal to the number of features in the dataset. Thus, each neuron in the input layer represents a feature from the dataset. Hidden layers are the intermediary layers of the neural network. Each layer takes the input from a previous layer, performs the calculations and passes the output onto the next layer.

Before the output from the neurons is passed on, an activation function is applied to it. This activation function makes sure that no individual neuron in a particular layer dominates the overall output of that layer. It restricts the output of a neuron within a

particular range, applying a uniform range for the whole layer. This range depends on the type of activation function used, some of which are listed below

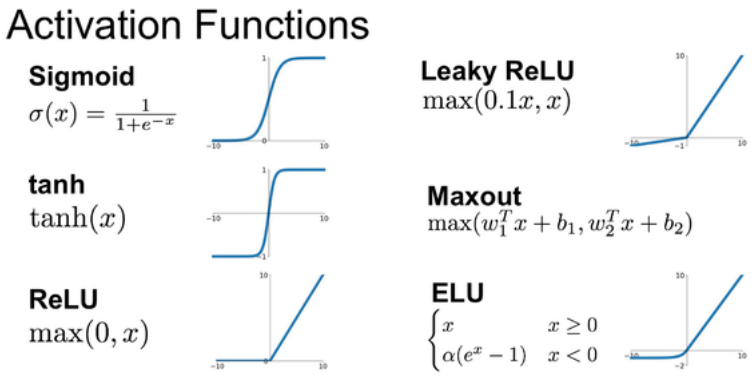

Figure 3.8: The different activation functions used in a neural network. Ng (2016*a*)

The output layer is the final layer of the neural network. The number of neurons in the output layer depend on the kind of problem being worked on. A regression problem will require a single neuron while a binary and k-class classification problem would require two and k neurons respectively. Since the output layer is the last layer, there is no bias term associated with the neurons of this layer.

Now, during the training of the neural network, the model alternates between forward propagation and backward propagation. Forward propagation is the normal process of going though the neural network layers from left to right. The input passes through the input layer first, gets processed in the hidden layers next and produces the output into the output layer.

A cost function is now used to calculate the accuracy of the output. Cost functions are error functions which quantify the difference between the predicted values and actual values. After one full forward propagation step for the entire dataset, backward propagation begins.

Backpropagation is done to update the weights of the individual neurons such that the predicted output is closer to the target output. The following steps are repeated for a given number of epochs till the error is minimum:

- Calculate the change in total error with respect to the result
- Calculate the change in result with respect to its sum
- Calculate the change in sum with respect to the weights

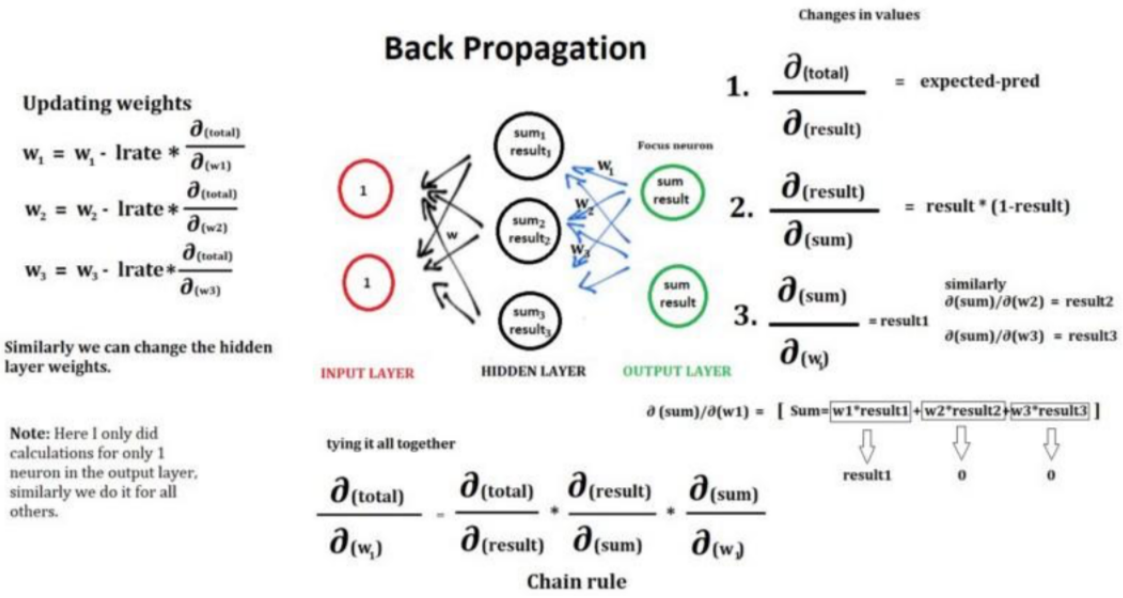

Figure 3.9: Mathematical representation of backpropagation calculations. Ng (2016*b*)

Thus, Neural Networks is a complex black-box model which is extremely competent in working with small dataset. It can capture complex inter-feature correlation that some of the simpler models are not able to. This makes Neural Networks an important cog in the wheel of the ensemble deep learning model. Haykin (1998)

3.6 Random Forest regressor

The random forest model is a type of additive model that makes predictions by combining decisions from a sequence of base models. More formally we can write this class of models as:

$$g(x) = f_0(x) + f_!(x) + f_2(x) + ...$$

where the final model g is the sum of simple base models f_i. Here, each base classifier is a simple decision tree. This broad technique of using multiple models to obtain better predictive performance is called model ensembling. In random forests, all the base models are constructed independently using a different subsample of the data. The subsample size is always the same as the original input sample size but the samples are drawn with replacement if bootstrap=True.

The use of multiple decision trees to create a mega-model is known as **Bootstrap Aggregation** or **bagging**. Bagging involves training of each individual decision tree

with a different subsample of dataset while the subsampling is done with replacement.

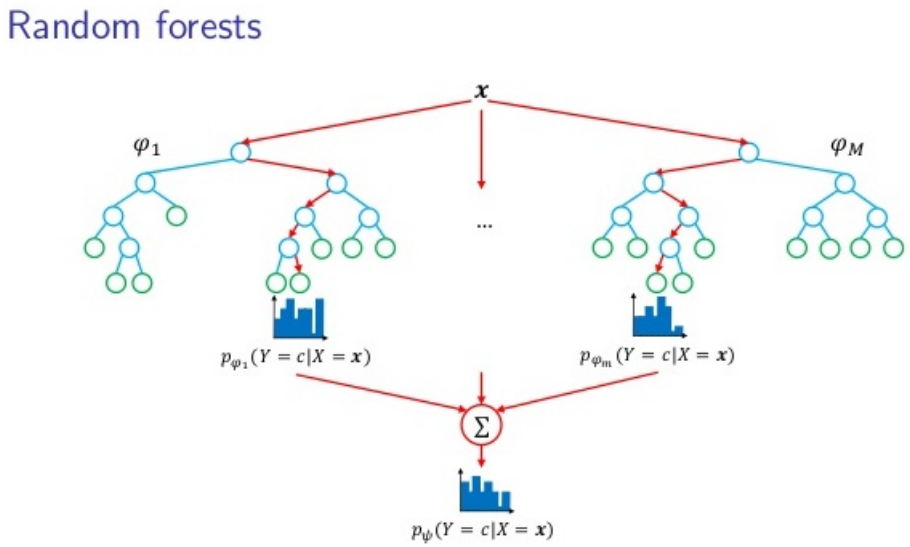

Figure 3.10: Simple illustration of Random forest model. Breiman (2001)

The final output for the random forest model is decided by taking a weighted average via a voting mechanism to rank the decision trees in the ensemble. This ensures the variance of the original dataset is accounted for along with reducing over-fitting in the model by taking the results from poorly fitted decision trees as well.

An ensemble method or ensemble learning algorithm consists of aggregating multiple outputs made by a diverse set of predictors to obtain better results. Formally, based on a set of ”weak” learners we are trying to use a ”strong” learner for our model. The purpose of using ensemble methods is to average out the outcome of individual predictions by diversifying the set of predictors, thus lowering the variance and to arrive at a powerful prediction model that reduces over-fitting of our training set.

Random forest is extremely efficient at handling tabular data with categorical features having fewer than hundred of categories. This feature can be numerical features or categorical features. if the feature is extremely sparse, then it needs to be pre-processed before being fed into the Random forest model.

Random forest is also very efficient at capturing the non-linear interactions between correlated features. This makes it ideal to work with small database. With the decision trees being transparent and easy to interpret, random forest is also a white-box model. Breiman (2001)

3.7 Gradient Boosting regressor

Gradient boosting is quite similar to random forest regressor in that the former is also an ensemble of ”weak” learners, most often decision trees, This collection of decision trees are assigned individual weights and the final output from the gradient boosting regressor is a weighted average of individual outputs from these decision trees.

The big point of difference between the random forest and gradient boosting regressor is the flexibility to choose the loss function in the latter. Random forest has a fixed loss function, known as *”Gini Impurity”*.

$$I_G(p) = \sum_{i=1}^{J} p_i \sum_{k \neq i} p_k = 1 - \sum_{i=1}^{J} p_i^2$$

Meanwhile, gradient boosting allows the developer to choose any differentiable loss function to optimize the model. At each stage or level, the decision tree is fit using the negative gradient of the loss function. Some of the loss functions which can be used are:

- Sum of least squares
- Least absolute deviation
- A combination of *Sum of least squares* and *Least absolute deviation*

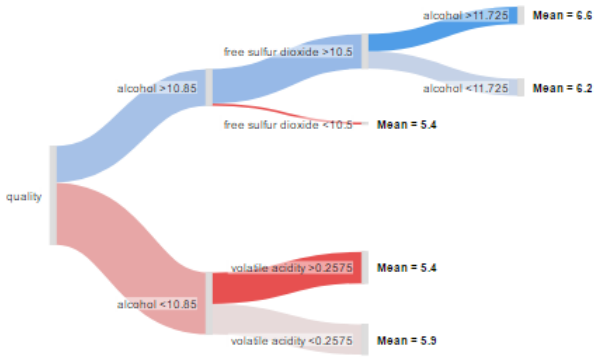

Figure 3.11: Gradient Boosting regressor explained using a Sankey diagram.

Apart from the above loss functions, custom made loss functions which are differentiable can also be applied to optimize the gradient boosting regressor. GB regressor is a forward stage-wise additive model. In the current study, Sum of least squares is used as the loss function of choice. Friedman (2002)

3.8 ADA Boost regressor

ADA Boost are a unique ensemble boosting model which also uses decision trees to work on regression problems. But a key difference between ADA Boost and other boosting regressors is that it uses decision trees with just a single level. These single level decision trees are also known as *decision stumps*.

ADA Boost regressor is a meta-estimator, that begins by fitting a regressor on the original dataset and then fits additional copies of the regressor on the same dataset but where the weights of instances are adjusted according to the error of the current prediction. As such, subsequent regressors focus more on difficult cases.

ADA Boost can be used to boost the performance of any machine learning algorithm. They perform better than random chance for any classification problem and can easily transition into and perform at a satisfactory level for regression problems.

Weighted average is taken to get the final output from the ensemble of decision trees. The number of decision trees used depends on the improvement made by the addition of new learners. Once the model is set, we are left with a pool of weak learners, each with a stage value. Zhu *et al.* (2009)

3.9 LightGBM

LightGBM is a gradient boosting framework that uses tree based learning algorithms. It is designed to be distributed and efficient with faster training speed and higher efficiency, lower memory usage and better accuracy. It is based on decision tree algorithms. LightGBM brings with it the following advantages:

- Higher efficiency and quicker training speed
- Low memory usage
- Better accuracy than most models
- Support for parallel learning and GPU learning.
- It can handle huge dataset.

Many boosting model use pre-sort based algorithms for decision tree learning. It is a simple solution which is not very easy to optimize. LightGBM, on the other hand, uses a histogram-based model. This technique buckets continuous variables into discrete bins. The advantage of this is the reduction in memory usage and increased speed in training.

One more advantage LightGBM offers is it's innovative leaf-wise growth algorithm which counters the traditional level-wise growth algorithm. In this method, the leaf with the maximum delta loss grows first, which can lead to one branch of the tree growing non-uniformly when compared to the whole tree. This is contrary to the level-wise growth algorithm in which each branch in a tree grows at the same pace, one level at a time.

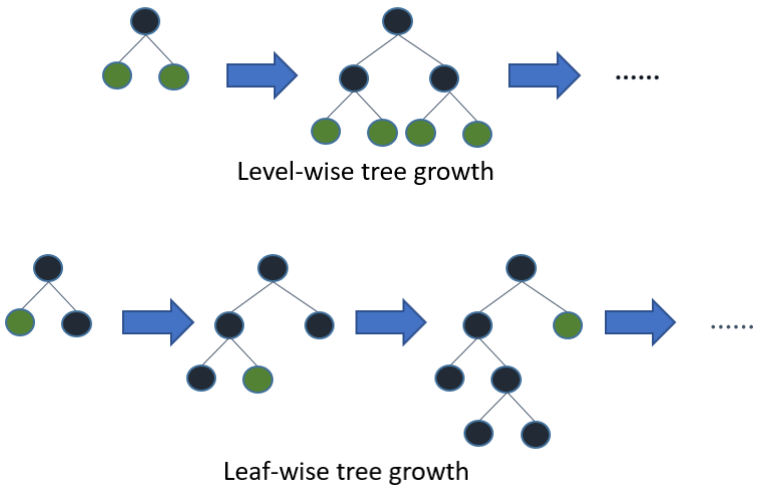

Figure 3.12: Comparing level-wise and leaf-wise growth algorithm. Ke *et al.* (2017)

One of the disadvantage of leaf-wise growth algorithm is over-fitting in the case of small dataset like in the current study. This is countered by fixing the depth of the tree, thus reducing this problem. Apart from this, parallel GPU processing and the application of state-of-the-art optimization algorithms makes LightGBM an attractive regression model to use. Ke *et al.* (2017)

3.10 XGBoost regressor

XGBoost stands for âĂIJExtreme Gradient BoostingâĂİ, where the term âĂIJGradient BoostingâĂİ originates from the paper *Greedy Function Approximation: A Gradient*

Boosting Machine, by *Friedman*. XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. It implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting that solve many data science problems in a fast and accurate way.

XGBoost works by having Classification And Regression Trees (CART) as it's weak learners. They classify the data-points into appropriate categories and then go about scoring them for regression problems. The basic working of XGBoost is pretty much similar to other boosting algorithms in which a loss function is defined along with regularization, training dataset is passed through the model with the aim of minimizing the loss.

XGBoost pushes your machine to its limit, in search for both accuracy and precision. It is currently one of the best boosting algorithm, one which identifies and captures the complex inter-feature correlation and preserves the variance of the dataset. It also reduces over-fitting while giving very high accuracy values. Chen and Guestrin (2016)

CHAPTER 4

RESULTS AND DISCUSSIONS

In this chapter, the steps taken to prepare the data, visualize the data, understand the data and the results of running machine learning models on this data are summarized. The first step taken on the dataset is normalization and scaling.

4.1 Data Exploration

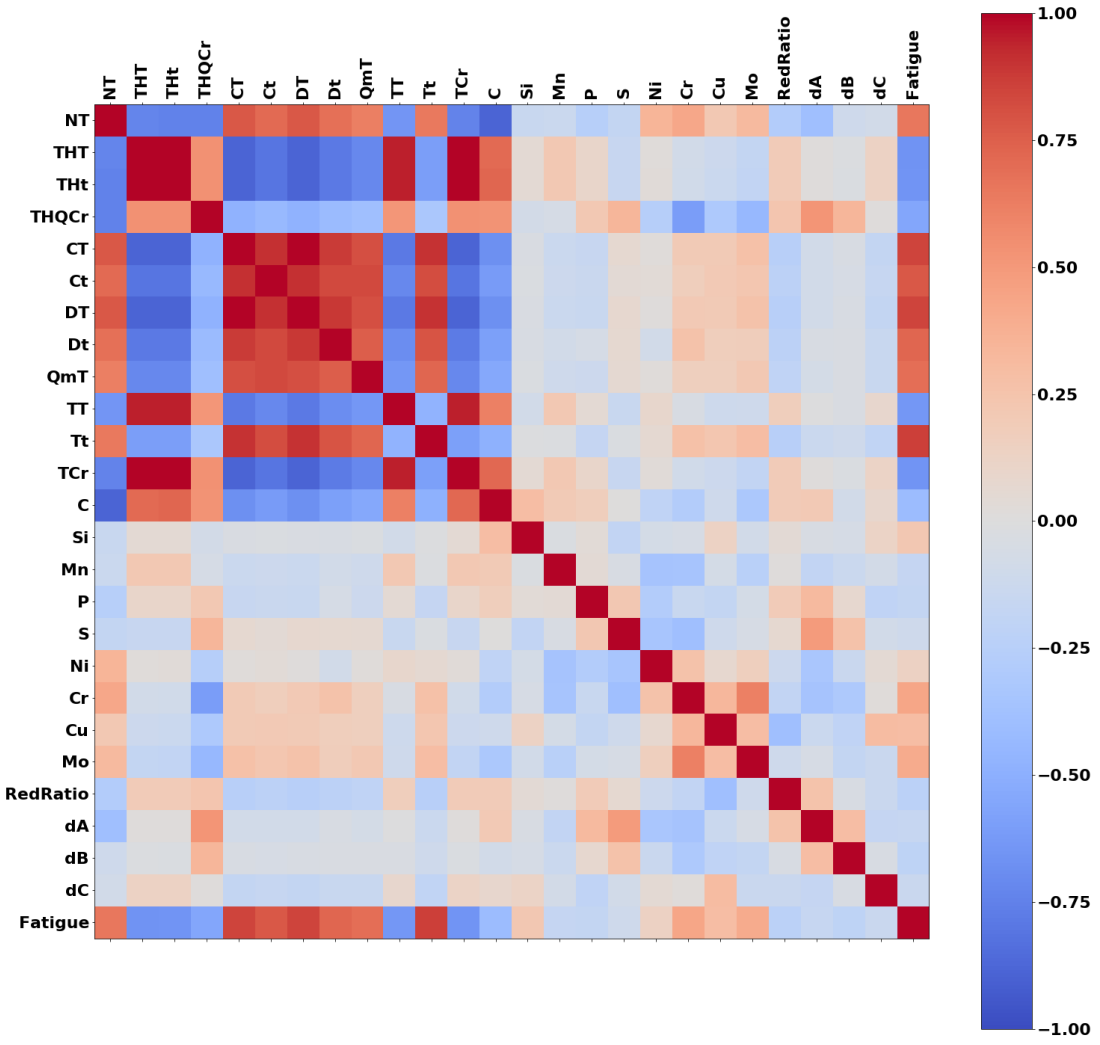

Figure 4.1: Heat map for the Pearson correlation coefficient.

The values tabulated for these features varies significantly in range, from 0.001 to 1000s units for chemical composition and Red Ratio respectively. Thus, it is important to normalize the dataset by bringing all the features into a common range.

Before doing this, the column *'Sl. No.'* has been dropped, reducing the number of columns to 26 from 27. Then, Standard Scaling is used to bring the values of these features to within the range of [-1, 1].

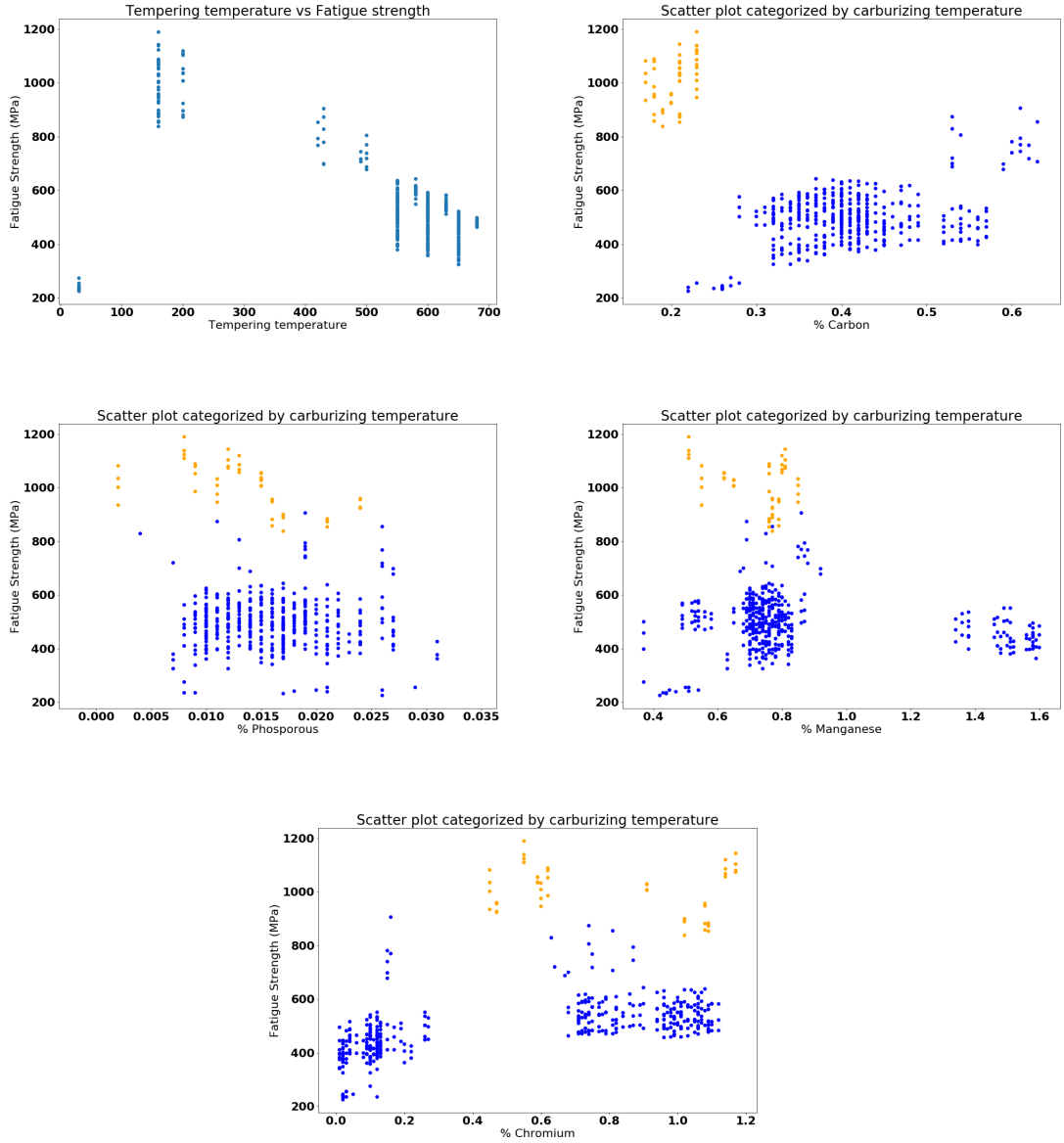

Figure 4.2: Data exploration using colour coded scatter plots.

Pearson correlation coefficient is then used to look at the correlation between different features present in the dataset. This is followed by data exploration using a combination of feature variables and target variables as show.

Some really interesting trends can be observed. When Fatigue Strength was plotted against the Tempering temperature, there were clear clusters visible, indicating that fatigue strength had a strong correlation with tempering temperature. This can also be confirmed from the Pearson correlation heat map, which has a mildly dark blue colour shade for fatigue strength vs tempering temperature, indicating strong negative coorelation between the two variables.

Apart from this, the chemical composition of the material showed strong correlation with the carburization temperature. Two clear clusters, colour coded using yellow and blue colour can be observed. Also observed are clear clusters for certain values of *Manganese* and *Chromium*. This, though, is not observed in the Pearson correlation coefficient heat map, making it an interesting observation.

4.1.1 Univariate Analysis

Univariate analysis examines the relationship of each feature with the target variable individually and then ranks them. This can be measured using Pearson coefficient, Maximal Information Coefficient or distance correlation.

Pearson Coefficient is a measure of the correlation between two variables. It has a value between -1 and +1 with -1 being perfect negative correlation and +1 being perfect positive correlation. Pearson correlation of 0 does not mean the variables are independent.

$$\rho(x, y) = \frac{cov(X, Y)}{\rho_x \rho_y}$$

Maximal Information Coefficient (MIC) is a measures of mutual dependence between two variables. MIC gives a score in bits which is not normalized. It is also inconvenient to compute it for continuous variables in general the variables need to be discretized by binning, but the mutual information score can be quite sensitive to bin selection.

$$I(X, Y) = \sum \sum p(x, y) log[\frac{p[x, y]}{p(x)p(y)}]$$

Distance correlation is a measure of dependence between two paired random vectors of arbitrary, not necessarily equal, dimension. Thus, distance correlation measures both linear and nonlinear association between two random variables or random vectors. The top five features according to the Univariate analysis for feature ranking are:

Features	Scores
Carburization Temperature	0.3067
Diffusion Temperature	0.2872
Carburization time	0.1976
Through Hardening Temperature	0.0529
Diffusion time	0.0253

Table 4.1: Feature ranking of the dataset based on Univariate analysis.

4.1.2 Recursive feature elimination

Given an external estimator that assigns weights to features, the goal of recursive feature elimination (RFE) is to select features by recursively considering smaller and smaller sets of features. First, the estimator is trained on the initial set of features and the importance of each feature is obtained. Then, the least important features are pruned from current set of features. That procedure is recursively repeated on the pruned set until the desired number of features to select is eventually reached.

The five features selected using Recursive Feature Elimination, in no particular order, are:

- Cooling rate for Through Hardening
- % Silicon
- % Manganese
- % Nickle
- % Chromium

4.1.3 Principal Component Analysis

Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a

set of values of linearly uncorrelated variables called principal components. This transformation is defined in such a way that the first principal component has the largest possible variance, and each succeeding component in turn has the highest variance possible under the constraint that it is orthogonal to the preceding components.

The resulting vectors are an uncorrelated orthogonal basis set. PCA is sensitive to the relative scaling of the original variables. For the current dataset, PCA was done by reducing the number of features down to two from twenty-five. Principal Component Analysis helps identify certain clusters or patterns which are hidden due to the curse of dimensionality.

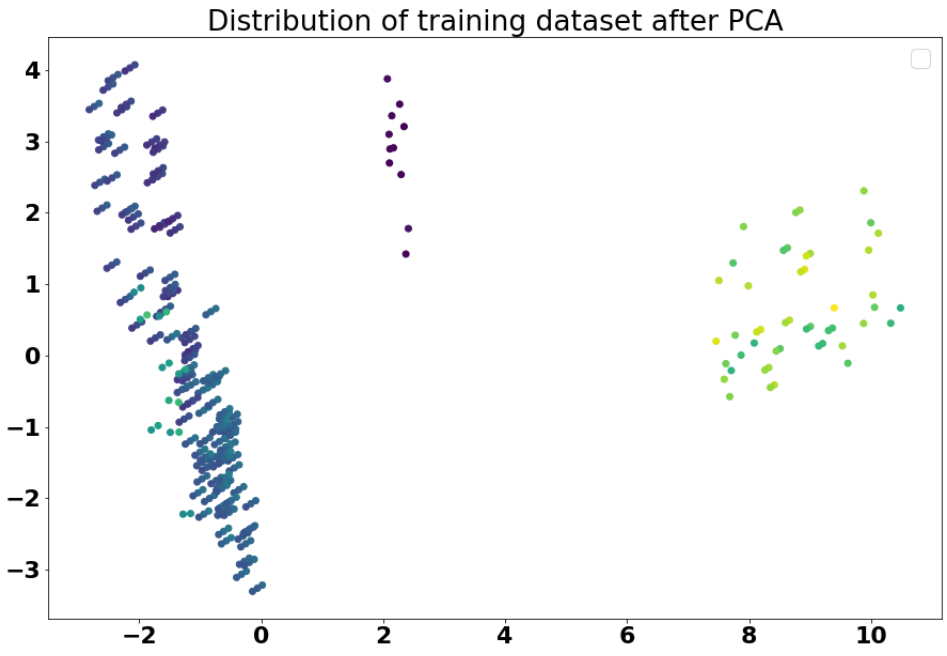

Figure 4.3: Principal Component Analysis of the fatigue strength dataset.

4.2 Modelling results

After exploring the dataset for patterns and correlations, the next step is to act upon the acquired information by running it through relevant models. In this section, the results obtained from the 10 chosen models are summarized. To look at the full code, navigate to the Appendix.

4.2.1 SVM Regressor

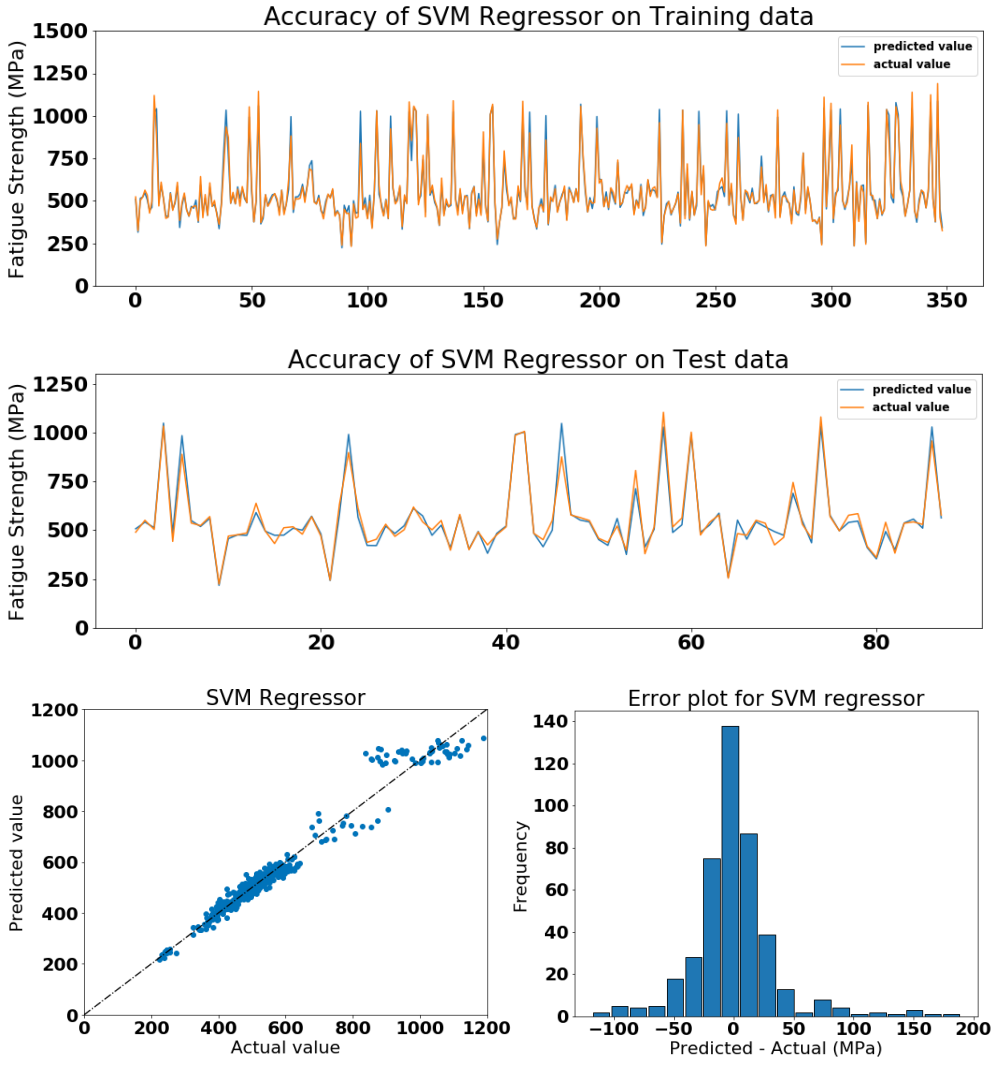

Figure 4.4: Results obtained from SVM Regressor.

Accuracy of SVM Regressor	$r^2 score$
Training Data	0.964832
Test Data	0.956210

SVM regressor manages to fit the training dataset to the tune of 96.5% accuracy with marginal reduction to 95.6% on test dataset. Thus, SVM regressor is a good start to fitting the fatigue strength dataset with reasonable accuracy.

4.2.2 Linear Regression

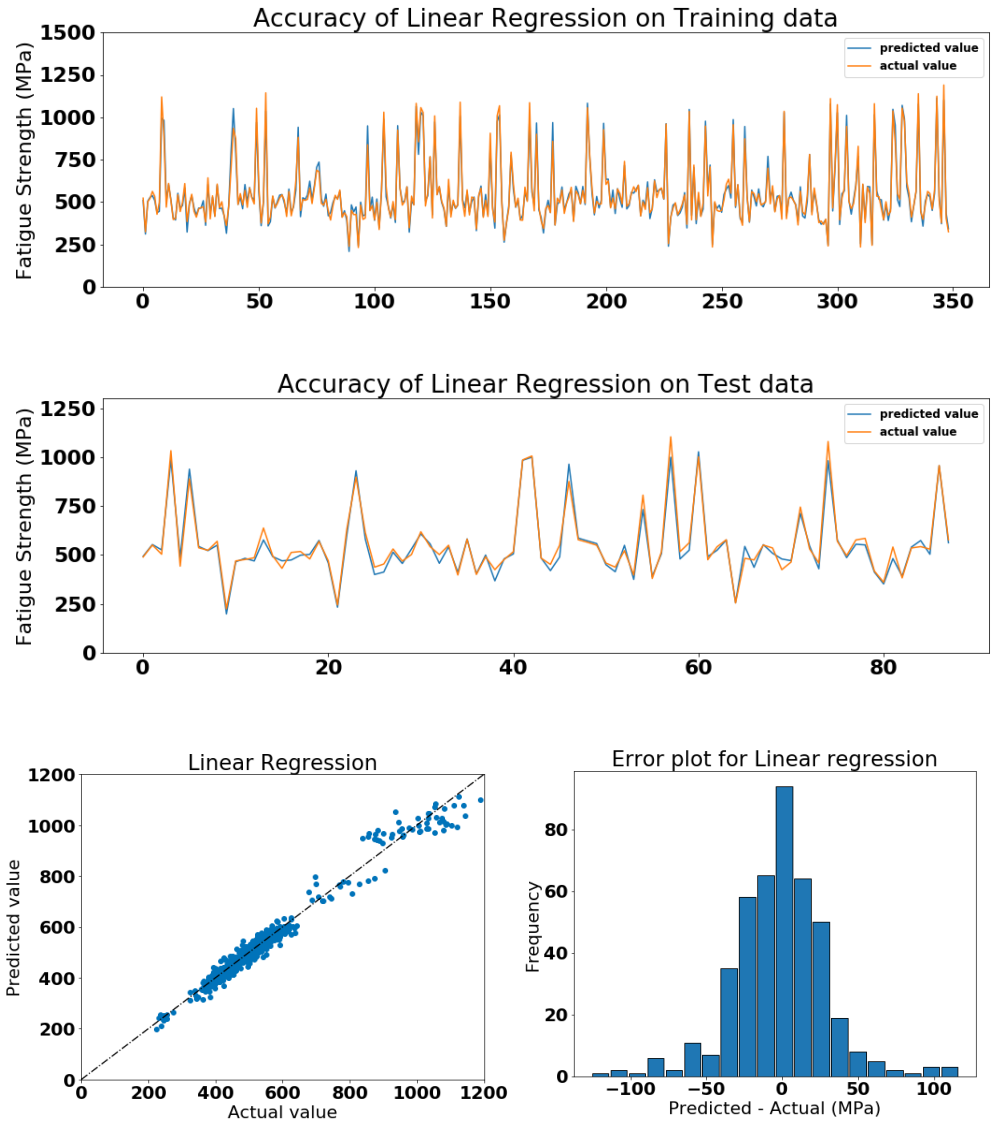

Figure 4.5: Results obtained from Linear Regression.

Accuracy of Linear Regressor	$r^2 score$
Training Data	0.972629
Test Data	0.966013

Linear regression, admittedly a simple and linear model, performs extremely well with an accuracy of 97.3% on training data with marginal reduction to 96.6% on test data. Thus, it shows that the dataset can be fitted reasonably well using a linear curve.

4.2.3 Polynomial Regression

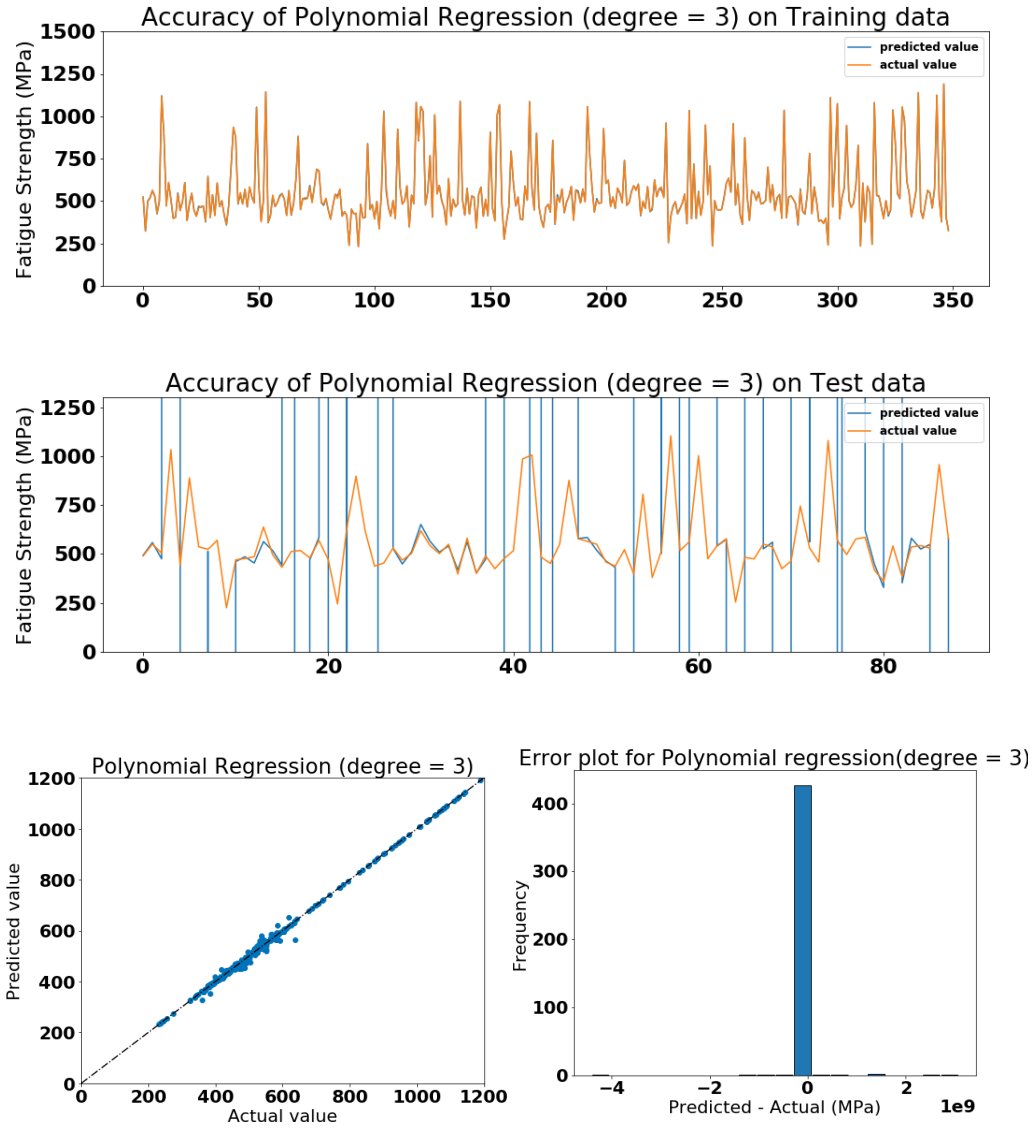

Figure 4.6: Results obtained from Polynomial Regression.

Accuracy of Polynomial Regressor	$r^2 score$
Training Data	0.972629
Test Data	0.966013

A simple polynomial regressor of degree = 3 is guilty of poor performance due to over-fitting. This is visible in the results for test dataset. Thus, it is a poor model to fit a small dataset.

4.2.4 Decision Tree Regression

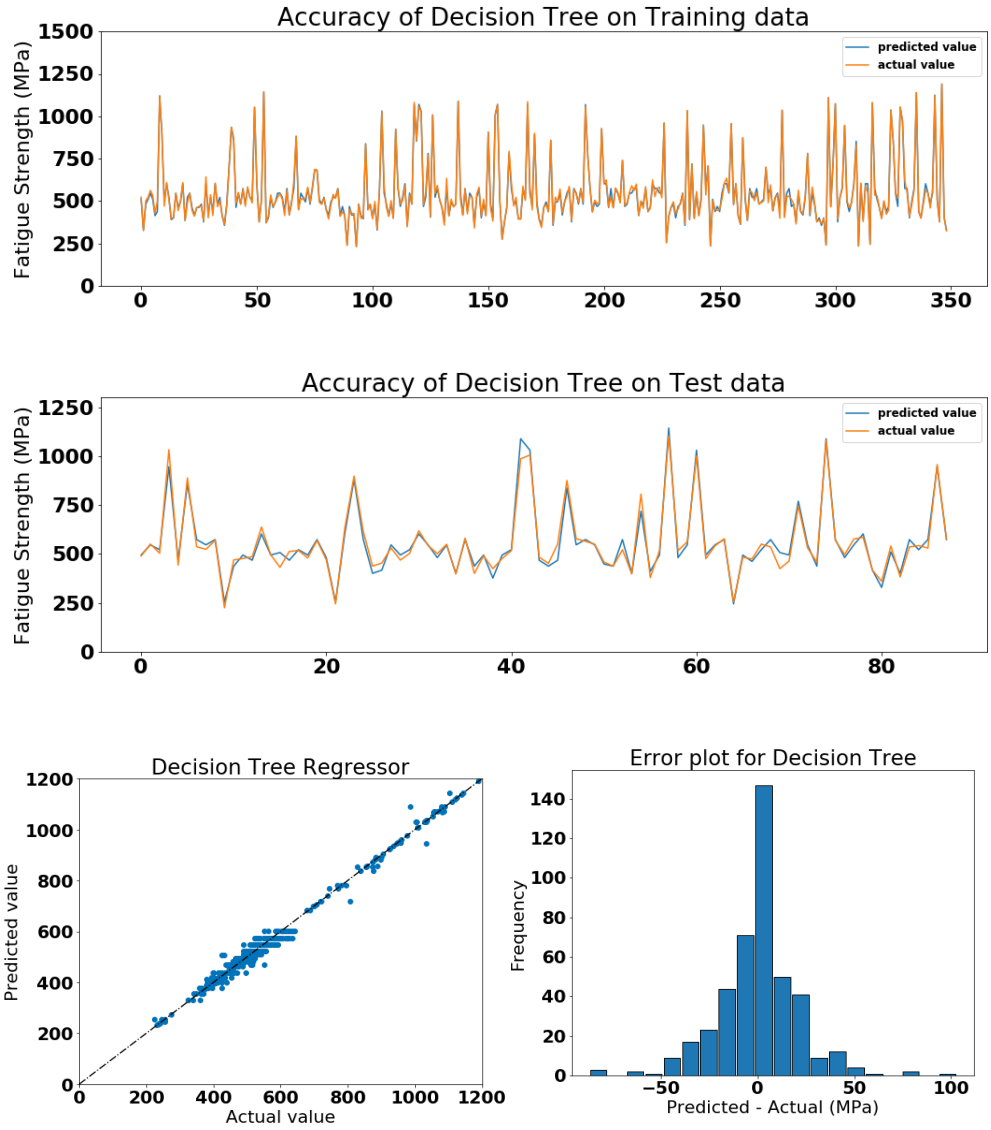

Figure 4.7: Results obtained from Decision Tree Regression.

Accuracy of Decision Tree Regressor	$r^2 score$
Training Data	0.992247
Test Data	0.966527

Decision Tree regressor shows extremely high accuracy on training dataset (99.2%) which drops significantly to 96.6% on the test dataset. It has very few high error points as seen in the scatter plot, thus making it a reliable model.

4.2.5 Neural Network

Layer (type)	Output Shape	# Parameters
Layer_1 (Dense)	(None, 512)	13132
Layer_2 (Dense)	(None, 256)	131328
Layer_3 (Dense)	(None, 128)	32,896
Layer_4 (Dense)	(None,64)	8256
Layer_5 (Dense)	(None, 01)	65

Table 4.2: Architecture of the Neural Network

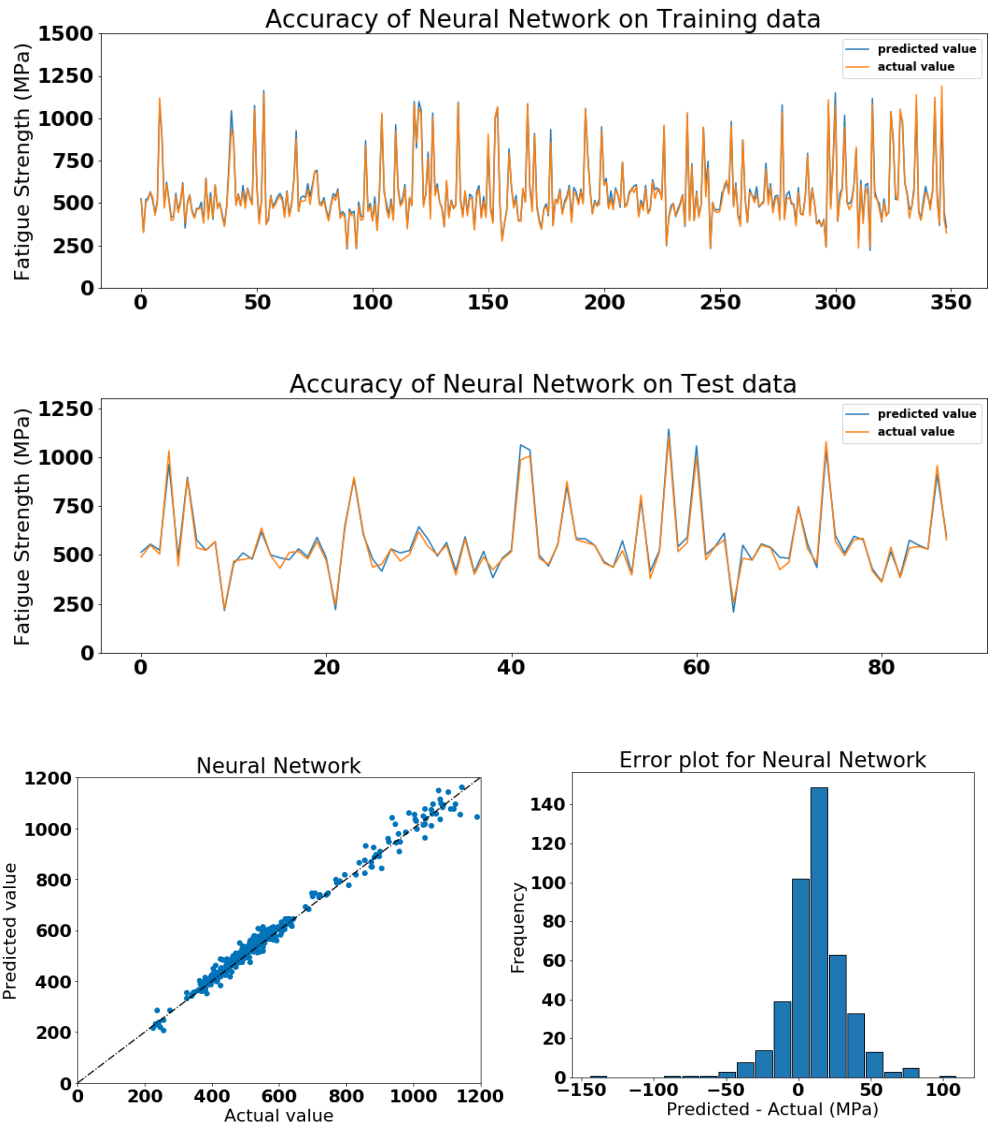

Figure 4.8: Results obtained from Neural Network.

Accuracy of Neural Network	$r^2 score$
Training Data	0.983781
Test Data	0.974342

Neural Network is one of the best models for the current dataset, with extremely high accuracy on training dataset (98.4%) with a marginal drop to 97.4% on test dataset. It is thus able to capture the complex inter-relations between features without over-fitting on the training dataset.

4.2.6 Random Forest Regressor

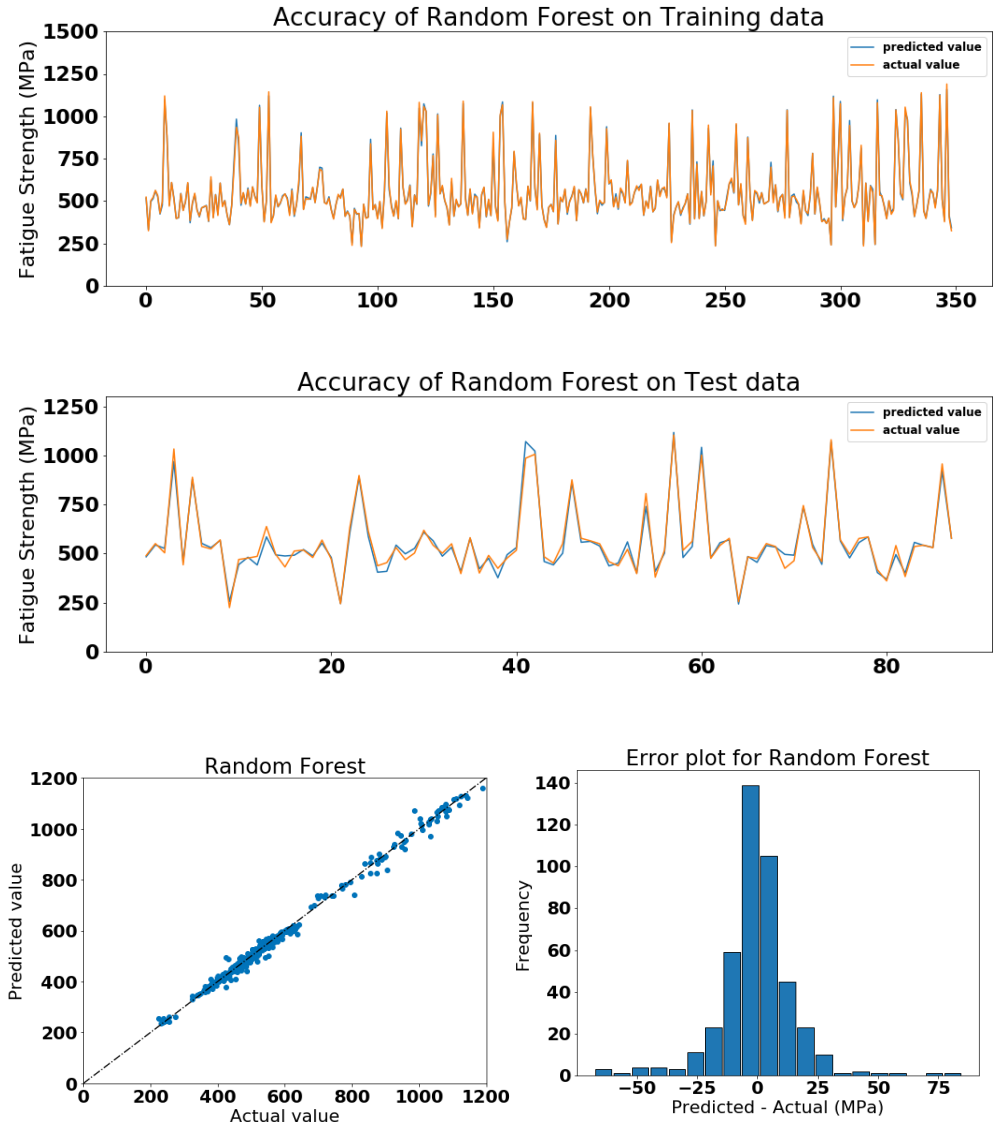

Figure 4.9: Results obtained from Random Forest Regressor.

Accuracy of Random Forest Regression	$r^2 score$
Training Data	0.996656
Test Data	0.977406

Random Forest regressor, just like Neural Networks, is extremely accurate on training dataset (99.7%) which dips marginally to 97.7% on test dataset. The scatter plot and

error plot show just how good Random Forest model is, with very small error values recorded for majority of data-points

4.2.7 Gradient Boosting Regressor

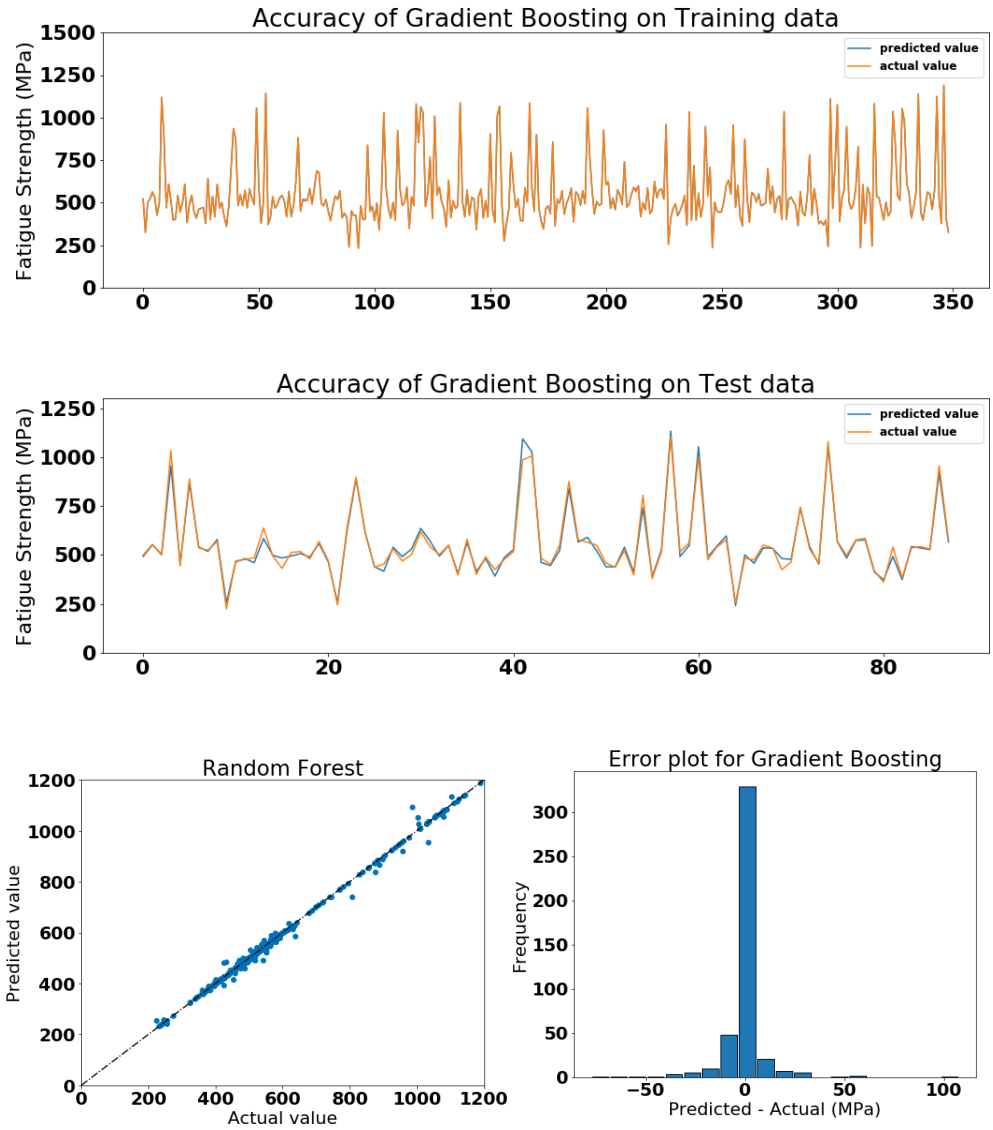

Figure 4.10: Results obtained from Gradient Boosting Regressor.

Accuracy of Gradient Boosting Regression	$r^2 score$
Training Data	0.999802
Test Data	0.979231

Gradient Boosting Regressor also follows similar trends but with a minor difference. The predictions on test data shows minor deviation from original values for almost all the predictions, with the error plot backing this observation as more than 300 data-points have infinitesimal error.

4.2.8 ADA Boost Regressor

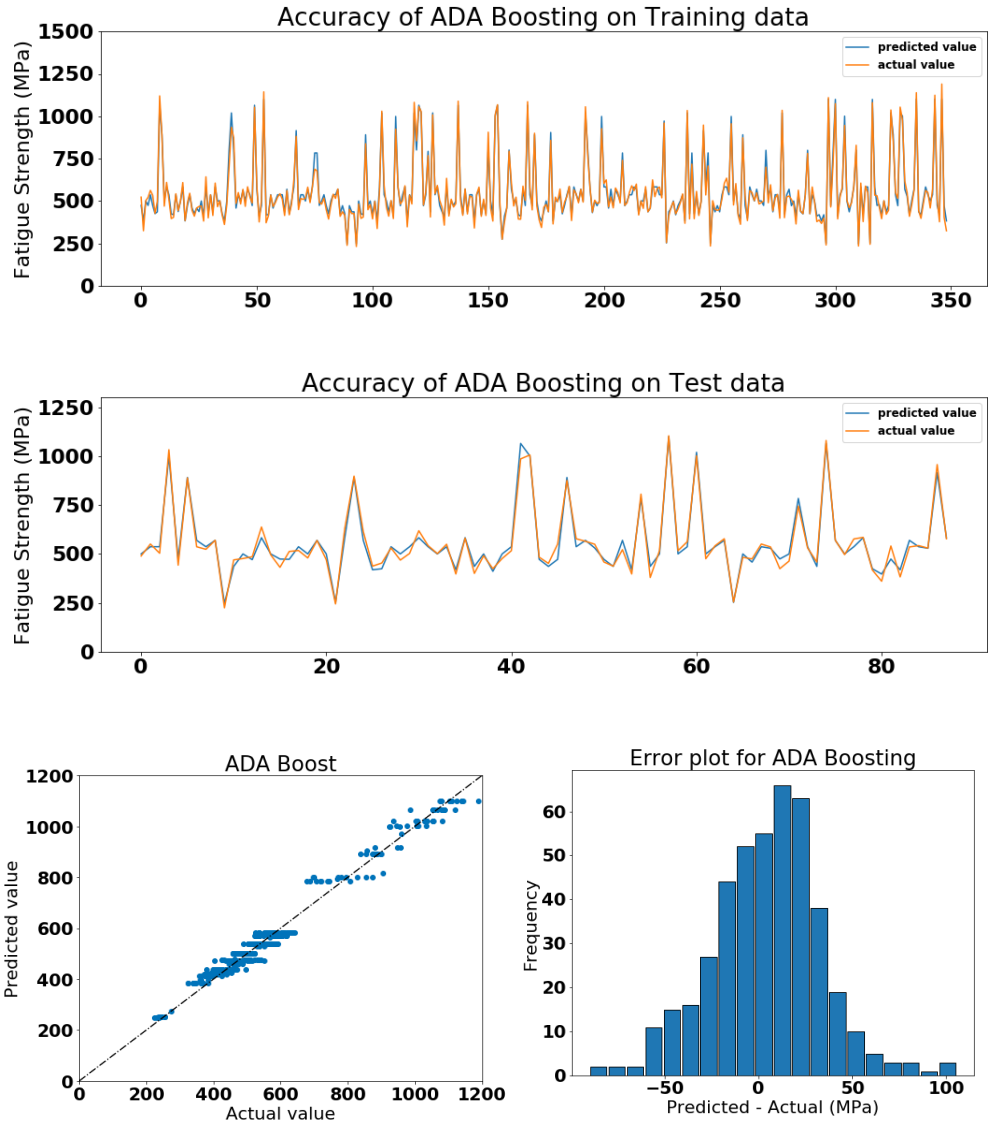

Figure 4.11: Results obtained from ADA Boost Regressor.

Accuracy of ADA Boost Regression	$r^2 score$
Training Data	0.974254
Test Data	0.974501

ADA Boost is one of the most consistent model, with very little to separate the accuracy for training dataset (97.43%) and test dataset (97.45%). But a close analysis of its scatter plot and error plot shows erratic predictions for certain data-points, reducing the reliability of the model.

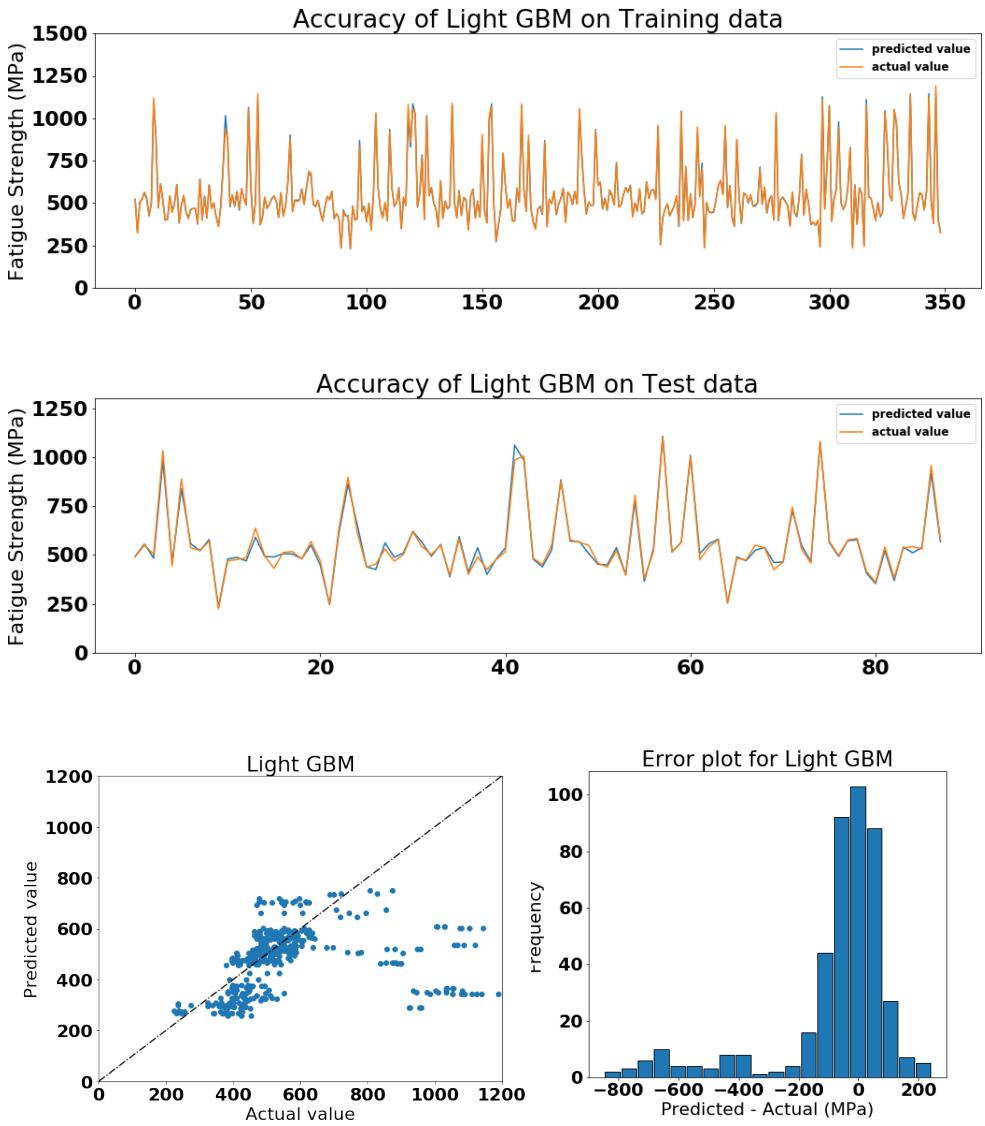

Figure 4.12: Results obtained from LightGBM Regressor.

Accuracy of LightGBM Regression	$r^2 score$
Training Data	0.997507
Test Data	0.984731

4.2.9 LightGBM Regressor

LightGBM regressor follows the pattern with very high accuracy on training dataset (99.7%) which drops marginally to 98.5% for test dataset. But it does show high error values for some data-points, as is visible from the scatter plot and the error plot. This reduces the dependability of LightGBM's predicted value.

4.2.10 XGBoost Regressor

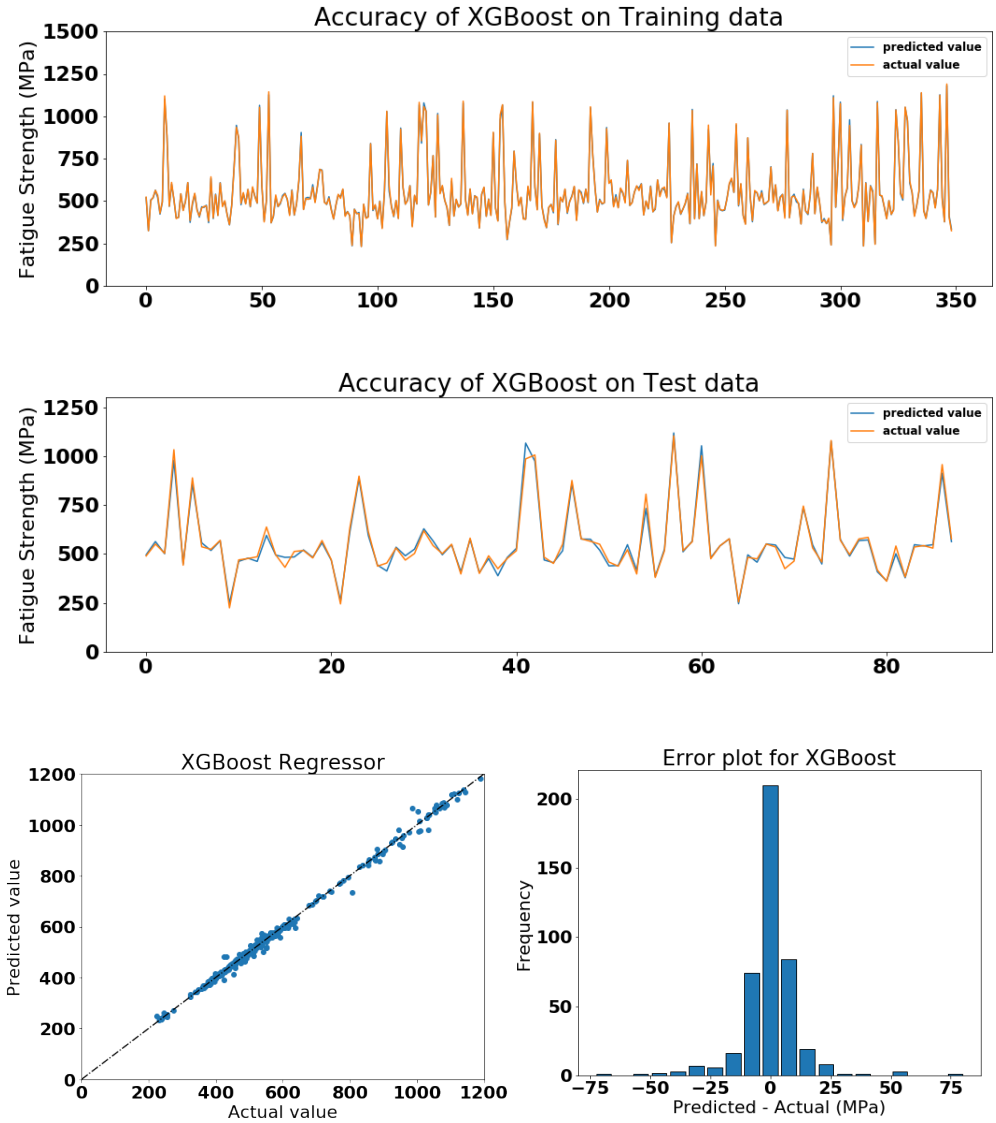

Figure 4.13: Results obtained from XGBoost Regressor.

Accuracy of XGBoost Regression	$r^2 score$
Training Data	0.998625
Test Data	0.983021

XGBoost regressor works perfectly with high accuracy on training dataset (99.9%) which drops marginally to 98.3% for test dataset. Looking at the scatter plot, it is clear and obvious that XGBoost is one of the best models for the current dataset.

4.2.11 Ensemble Deep Learning model

As seen above, some models like **Decision Tree regressor, Neural Network, Random Forest** and **XGBoost regressor** show extremely good data fitting and extremely high test dataset accuracy. Meanwhile, models like LightGBM and ADA Boost regressor show high variance away from the actual value for certain data-points with infinitesimal error for the rest.

We thus have a variety of models, some which come close to over-fitting the dataset without actually doing so, with some showing high error values. The results, in terms of the accuracy, are shown in the table below:

Model name	$r^2 score$ for Training data	$r^2 score$ for Test data
SVM regressor	0.964832	0.956210
Linear regression	0.972629	0.966013
Polynomial regression	0.972629	0.966013
Decision Tree regressor	0.992247	0.966527
Neural Network	0.983781	0.974342
Random Forest regressor	0.996656	0.977406
Gradient Boosting regressor	0.999802	0.979231
ADA Boost regressor	0.974254	0.974501
LightGBM regressor	0.997507	0.984731
XGBoost regressor	0.998625	0.983021

Table 4.3: Accuracy of the ten models employed

The above scores are achieved when 25 features are used to train the model. But when the number of features are brought down to 5, the performance also drops down. The choice of the top 5 features are explained using the *feature_importance* function for different models. They have been tabulated below:

S. No.	Model	Rank 1	Rank 2	Rank 3	Rank 4	Rank 5
1	Decision Tree regressor	Dt	Cr	C	TT	NT
2	ADA Boost regressor	TT	Cr	Tt	Dt	Ct
3	Random Forest regressor	QmT	Ct	Cr	NT	Tt
4	LightGBM regressor	TT	C	Cr	P	Mn
5	XGBoost regressor	TT	C	Cr	Mn	P

Table 4.4: Feature ranking by some of the top performing models

When selecting the top five features, the following points were considered:

1. The *feature_importance_* rankings were used to select the top five features.
2. Temperature based features were given greater importance over time based features.
3. Importance was given to chemical composition over steelmaking process parameter.
4. The features selected were purely based on the mathematical and machine learning model outputs. No theoretical considerations were taken.

After considering the above points, the following five features were selected as the top five features from the 25 features dataset:

Tempering Temperature (TT)

Studies done by Haftirman *et al.* (2007) on *Thyssen 6582 steel* and by SUIMON *et al.* (2010) on *Ni-Cr-Mo steel* showed that for a particular range of tempering temperature, high cycle fatigue strength increased with increasing temperature. But once the temperature transitioned out of this range and into high temperature range, fatigue strength started showing inverse relations with the tempering temperature. This shows that tempering temperature directly affects the fatigue strength of a material.

% Carbon (C)

Study done by Tayanc *et al.* (2007) on dual-phased steel showed that, while as drawn steel did not make much movement when % carbon was change by a small margin, but heat treated steel showed high degree of fluctuation in the value of fatigue strength. The direction in which fatigue strength moved depended on the range in which % carbon of steel was. Thus, the choice of % carbon having an effect on fatigue strength is backed by experimental results.

% Chromium (Cr)

Experimental study done by Kvedaras *et al.* (2006) on *chromium-plated steel* showed a change in fatigue strength of steel as the % chromium present in it was varied. This

study was backed by Williams and Hammond (1954) whose study on *En25 (Ni-Cr-Mo) Steel* also backed the observation of fatigue strength's dependence on % chromium present in steel. Given it's high feature ranking, it has been selected as one of the five features, also due to the ease with which it can be measured with high accuracy.

% Manganese (Mn)

Experimental Study done by Farrara and H. Underwood (1990) on the effects of Manganese Phosphate to tensile and fatigue strength shows the correlation between the % Manganese present in the steel and fatigue strength of the material. Thus, the feature ranking of Manganese as one of the important and highly correlated feature with fatigue strength is theoretically backed up.

% Phosphorus (P)

Experimental study done by an ensemble of Castro *et al.* (2010) on SAE 5160 Steel showed some correlation between % phosphorus content and fatigue strength. This study was backed up by S. Hyde *et al.* (1994) who studied gas carburized 4320 steel and the effect of phosphorus on it. The small changes observed conclusively prove a good correlation between the amount of phosphorus, which is also a chemical property of the microstructure, and the fatigue strength of the material.

Thus, **Tempering Temperature, % Carbon, % Chromium, % Manganese** and **% Phosphorus** are the top five features. They are selected because four out of the five features are chemical properties of the microstructure, which are easy to measure at high precision. The fifth feature, Tempering Temperature, also shows high correlation with fatigue strength as seen in the Pearson correlation heat map and feature ranking table.

The next step in the process is to figure out which model to choose to fit a five-features dataset. To make that decision, all the ten models are trained using just these five features and their new accuracy are tabulated in **Table** 4.5

From the above table, simplistic models like SVM regressor and Linear regression

Model name	$r^2 score$ for Training data	$r^2 score$ for Test data
SVM regressor	0.497909	0.209385
Linear regression	0.617179	0.528545
Polynomial regression	0.976667	0.0.961134
Decision Tree regressor	0.989510	0.962687
Neural Network	0.945696	0.937432
Random Forest regressor	0.991512	0.963090
Gradient Boosting regressor	0.997648	0.976408
ADA Boost regressor	0.959724	0.946410
LightGBM regressor	0.994032	0.930288
XGBoost regressor	0.996073	0.979749

Table 4.5: Model accuracy for just five features.

models are immediately eliminated. Meanwhile, Polynomial regression already showed that it had over-fitting issues with the original dataset. Thus, it is also taken out of consideration. Also, from previous summary of all the models, none of the models, standalone, are able to capture the complex inter-feature coorelation while preserving the variance of the dataset at a very high accuracy.

This leads to the choice of making an ensemble machine learning model. In an ensemble model, more than one machine learning model, usually models which differ in the trends they have the ability to capture, are chosen to form an ensemble model. This leads to the ensemble model performing an extremely good job of fitting the current dataset.

Thus, from the remaining seven models, more than one models need to be chosen to form the ensemble machine learning model. **Neural Networks, Decision tree regressor, XGBoost regressor** and **Random Forest regressor** perform extremely well on both the dataset - one with twenty-five features and the other with five features. But since Neural Network and Random Forest regressor are extremely similar in capturing similar trends from the dataset, one among the two, specifically Neural Network over Random Forest regressor, is chosen.

The key to a highly accurate and ensemble model is to have a good variety of models and adding weak learners to learn from their mistakes. Keeping this in mind, **ADA Boost regressor** and **LightGBM regressor** are also chosen. Both these models show high training data accuracy but falter when predicting on training dataset. They add a good mix to the ensemble ”Deep Learning” model.

The choice of leaving Gradient Boosting regressor out of the ensemble was down to the fact that Gradient Boosting regressor and XGBoost regressor are almost similar models, with XGBoost trumping the former in terms of accuracy and capture of variance. Thus, we now have the five models which make the **Ensemble Deep Learning model**.

Ensemble Deep Learning model
Decision Tree regressor
Neural Network
ADA Boost regressor
LightGBM regressor
XGBoost regressor

Table 4.6: Models selected for the Ensemble Machine Learning model

This ensemble model is now called an *Ensemble Deep Learning Model* due to the presence of an Artificial Neural Network with more than one hidden layer, which makes it a deep learning model. The performance of this Ensemble deep learning model is summarized:

Accuracy of Ensemble deep Learning model	$r^2 score$
Training Data	0.989414
Test Data	0.975358

The accuracy and precision of the ensemble deep learning model almost mirrors that of individual Neural Network or XGBoost regressor for twenty-five features, thus making it a very good fit for the dataset. Looking at the scatter plot, the predicted values are hovering very close to the $y = x$ line, meaning that the predicted values are extremely close to the actual values.

The error plot shows that there are some points which have high error values, but that is expected given only five features have been used to predict the fatigue strength. It can be observed that the error values are less than $\pm 25 MPa$. It should also be noted that more than 200 out of the 437 data points show error values close to 0. Thus, the ensemble deep learning model is the final choice for the given dataset and is used to build the fatigue strength calculator.

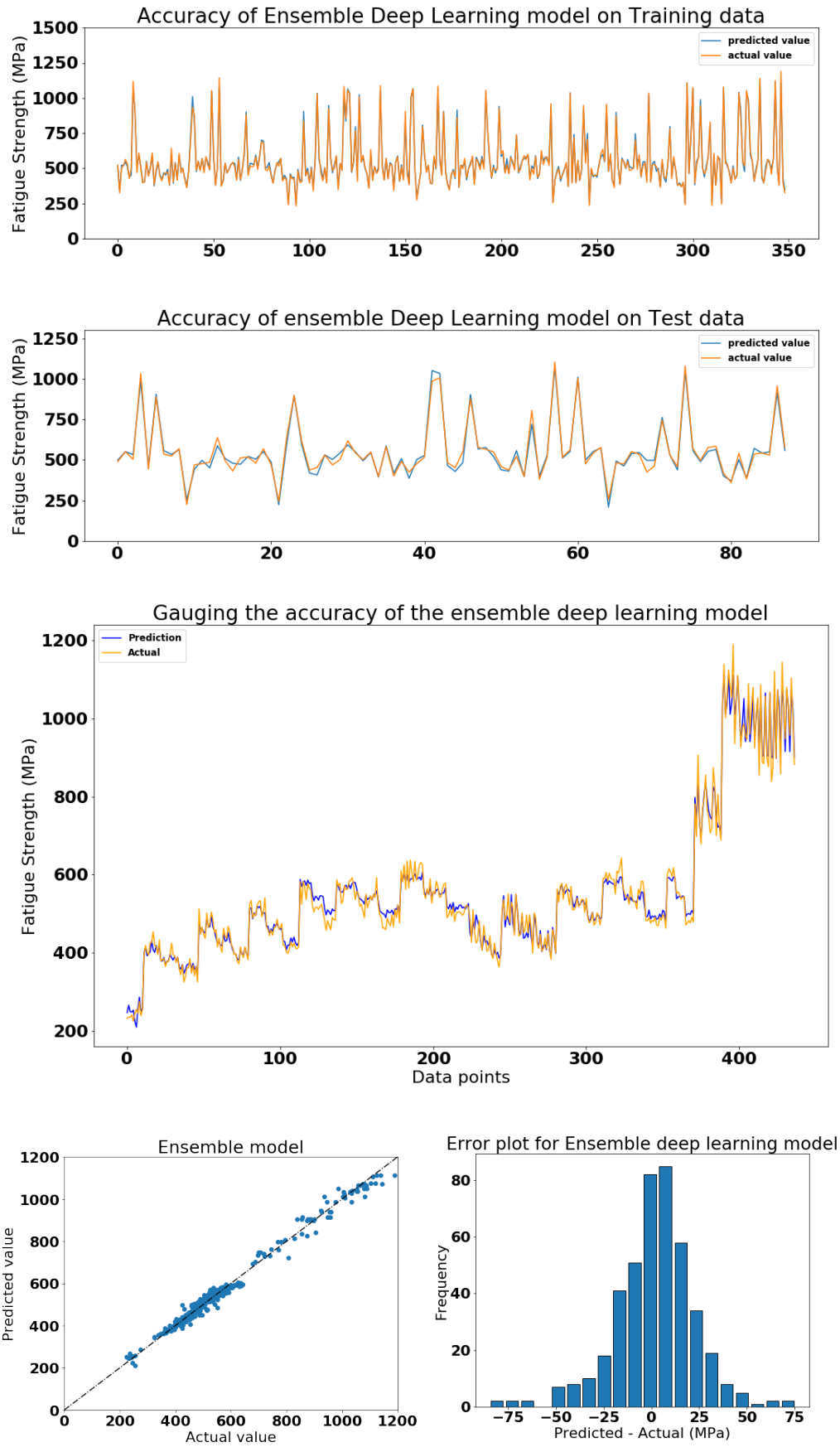

Figure 4.14: Performance of Ensemble deep learning model when fitting fatigue strength dataset with just five features.

CHAPTER 5

CONCLUSION

The aim of the current study was to use the fatigue strength dataset to build a fatigue strength calculator calculated using the least number of inputs possible. But with just 437 data-points, it was challenging to fit the dataset using a machine learning model with very high accuracy. The challenge was also to reduce the number of features used by this model to a number smaller than the total number of features present in the dataset.

This is why an ensemble deep learning model, consisting of some of the more advanced models like Multi-layered Artificial Neural Network were employed. While the accuracy of the ensemble model is very good, the small size of the dataset makes it tough to generalize outliers. This is why poor learners like LightGBM regressor and ADA Boost regressor were added to diversify the ensemble.

The number of features were reduced considerably from the twenty-five features originally present in the dataset to just five features, which is a significant add-on to the work done by Agrawal *et al.* (2014*b*) in the paper *Exploration of data science techniques to predict fatigue strength of steel from composition and processing parameters*.

The fatigue strength calculator is built by first compressing and converting the ensemble deep learning model into an executable *.pkl* file using the python library ”pickle”. This, along with an executable HTML code for the webpage is uploaded onto the ”**Heroku**” application.

Heroku executes the compressed model in the backend of the webpage while retrieving the input values from the webpage. It then runs the input values through the model and displays the output predicted by the model on the webpage. This webpage is free-for-use and open-for-all. A snapshot of the webpage is attached in the next page.

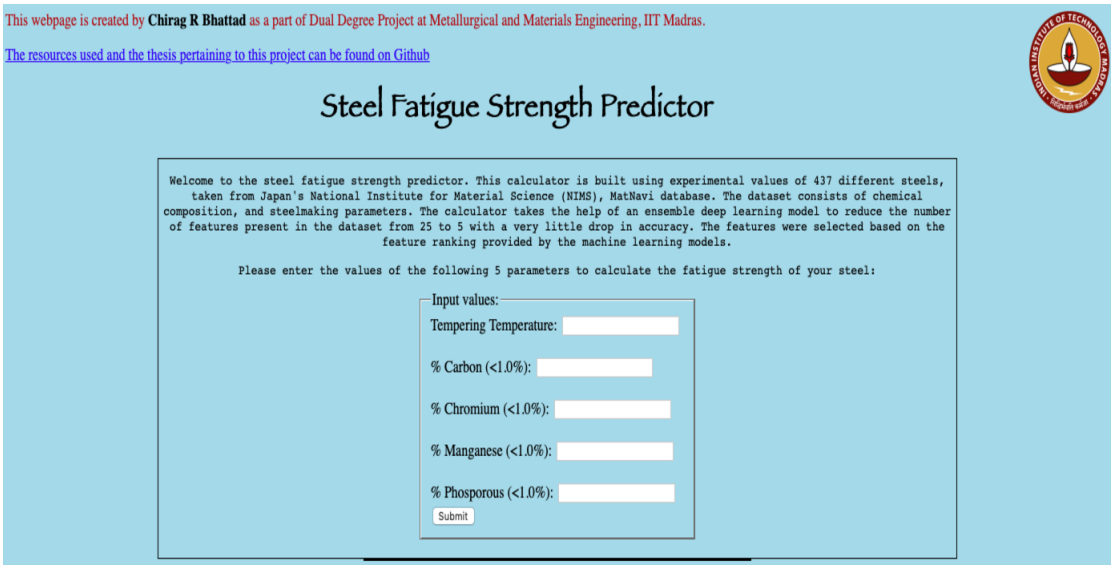

Figure 5.1: Snapshot of the fatigue strength calculator.

The dataset used and the proprietary code written for this study has been made open-source on the github link provided on the webpage. This is done to ensure more and more data-points are added to the dataset as and when more experiments/simulations are done. The hope is also to improve and increase the accuracy as and when better models are theorized.

APPENDIX A

CODE REPOSITORY

```
# Importing relevant libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import PolynomialFeatures
from sklearn.decomposition import PCA
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeRegressor
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import RFE
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import AdaBoostRegressor
import tensorflow as tf
import xgboost as xgb
import lightgbm as lgb
from sklearn.svm import SVR
from tensorflow import keras
from tensorflow.keras import layers
import keras
from keras.models import Sequential
from keras.callbacks import ModelCheckpoint
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasRegressor
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
```

```
from sklearn.pipeline import Pipeline
import pickle
import json
import requests
import statistics

# Loading the dataset and dropping the ''Serial No.'' column.
data = pd.read_excel("fatigue.xlsx")
data.drop(['Sl. No.'],axis=1, inplace=True)

# Customizing the look of plots generated by matplotlib.
font = {'family' : 'normal',
        'weight' : 'bold',
        'size'   : 22}
matplotlib.rc('font', **font)

#Plotting the Pearson correlation coefficient heatmap
corr = data.corr()
fig = plt.figure(figsize=(25,25))
ax = fig.add_subplot(111)
cax = ax.matshow(corr,cmap='coolwarm', vmin=-1, vmax=1)
fig.colorbar(cax)
ticks = np.arange(0,len(data.columns),1)
ax.set_xticks(ticks)
plt.xticks(rotation=90)
ax.set_yticks(ticks)
ax.set_xticklabels(data.columns)
ax.set_yticklabels(data.columns)
plt.show()

# Splitting the dataset into feature variable and target variable
feature = data.columns[:-1]
target = data.columns[-1]
X = data[feature]
y = data[target]

# Data visualization - Tempering Temperature vs fatigue Strength
plt.scatter(data['TT'], data['Fatigue'])
fig = plt.gcf()
fig.set_size_inches(15, 10)
```

```
plt.title('Tempering temperature vs Fatigue strength')
plt.xlabel('Tempering temperature')
plt.ylabel('Fatigue Strength (MPa)')

# Data visualization - % Carbon vs fatigue Strength
label = data['CT']
color= ['blue' if l == 30 else 'orange' for l in label]
fig = plt.gcf()
fig.set_size_inches(15, 10)
plt.scatter(data['C'], data['Fatigue'], color=color)
plt.title('Scatter plot categorized by carburizing temperature')
plt.xlabel('% Carbon')
plt.ylabel('Fatigue Strength (MPa)')

# Data visualization - % Phosphorus vs fatigue Strength
label = data['CT']
color= ['blue' if l == 30 else 'orange' for l in label]
fig = plt.gcf()
fig.set_size_inches(15, 10)
plt.scatter(data['P'], data['Fatigue'], color = color)
plt.title('Scatter plot categorized by carburizing temperature')
plt.xlabel('% Phosporous')
plt.ylabel('Fatigue Strength (MPa)')

# Data visualization - % Manganese vs fatigue Strength
label = data['CT']
color= ['blue' if l == 30 else 'orange' for l in label]
fig = plt.gcf()
fig.set_size_inches(15, 10)
plt.scatter(data['Mn'], data['Fatigue'], color = color)
plt.title('Scatter plot categorized by carburizing temperature')
plt.xlabel('% Manganese')
plt.ylabel('Fatigue Strength (MPa)')

# Data visualization - % Chromium vs fatigue Strength
label = data['CT']
color= ['blue' if l == 30 else 'orange' for l in label]
fig = plt.gcf()
fig.set_size_inches(15, 10)
plt.scatter(data['Cr'], data['Fatigue'], color = color)
```

```
plt.title('Scatter plot categorized by carburizing temperature')
plt.xlabel('% Chromium')
plt.ylabel('Fatigue Strength (MPa)')

# Univariate Analysis
test = SelectKBest(score_func=chi2, k=4)
fit = test.fit(X, y)
np.set_printoptions(precision=3)
print(fit.scores_/sum(fit.scores_))
features = fit.transform(X)
print(features[0:5,:])

# Recursive feature ranking
model = LogisticRegression()
rfe = RFE(model, 5)
fit = rfe.fit(X, y)
print("Num Features: %d"% fit.n_features_)
print("Selected Features: %s"% fit.support_)
print("Feature Ranking: %s"% fit.ranking_)

# Feature Scaling
sc = StandardScaler()
X = sc.fit_transform(X)

# Principal Component Analysis
pca = PCA(n_components=2, random_state=1)
principal_components = pca.fit_transform(X)
plt.scatter(principal_components[:,0],principal_components
[:,1], c = y)
plt.title("Distribution of training dataset after PCA")
fig = plt.gcf()
fig.set_size_inches(15, 10)

# Splitting the dataset into Training and Test dataset
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=1)

# SVM Regression
svreg = SVR(kernel='linear', verbose=1, C=15)
svreg.fit(X_train, y_train)
```

```
SVRtrain = r2_score(y_train, svreg.predict(X_train))
print("\nSVM Regressor Train data:", SVRtrain)
Y_pred_3 = svreg.predict(X_test)
SVRerror = r2_score(y_test, Y_pred_3)
print("SVM Regressor Test data:", SVRerror)

# Linear Regression
linreg = LinearRegression().fit(X_train,y_train)
linregression = linreg.score(X,y)
LTerror = r2_score(y_train, linreg.predict(X_train))
print("Linear Regression Training data:", LTerror)
Y_pred_1 = linreg.predict(X_test)
linerror = r2_score(y_test, Y_pred_1)
print("Linear Regression Test data:", linerror)

# Polynomial Regression
poly = PolynomialFeatures(degree = 3)
X_poly = poly.fit_transform(X)
poly.fit(X_poly, y)
X_train_poly = poly.fit_transform(X_train)
polreg = LinearRegression().fit(X_train_poly,y_train)
PTerror = r2_score(y_train, polreg.predict(X_train_poly))
print("Polynomial Regression Training data:", LTerror)
X_test_poly = poly.fit_transform(X_test)
Y_pred_16 = polreg.predict(X_test_poly)
polerror = r2_score(y_test, Y_pred_16)
print("Polynomial Regression Test data:", linerror)

# Decision Tree Regression
dtregr = DecisionTreeRegressor(max_depth=7, random_state=1)
dtregr.fit(X_train, y_train)
DTRtrain = r2_score(y_train, dtregr.predict(X_train))
print("Decision Tree Train data:", DTRtrain)
Y_pred_4 = dtregr.predict(X_test)
DTRerror = r2_score(y_test, Y_pred_4)
print("Decision Tree Test data:", DTRerror)

# Neural Network
NN_model = Sequential()

```

```
# The Input Layer :
NN_model.add(Dense(512, kernel_initializer='normal',
input_dim = 25, activation='relu'))

# The Hidden Layers :
NN_model.add(Dense(256, kernel_initializer='normal',
activation='relu'))
NN_model.add(Dense(128, kernel_initializer='normal',
activation='relu'))
NN_model.add(Dense(64, kernel_initializer='normal',
activation='relu'))

# The Output Layer :
NN_model.add(Dense(1, kernel_initializer='normal',
activation='linear'))

# Compile the network :
NN_model.compile(loss='mean_absolute_error',
optimizer='adam', metrics=['mean_absolute_error'])
NN_model.summary()

# Saving the weights of the trained Neural Network
checkpoint_name = 'Weights-{epoch:03d}--{val_loss:.5f}.hdf5'
checkpoint = ModelCheckpoint(checkpoint_name,
monitor='val_loss', verbose = 1, save_best_only
= True, mode ='auto')
callbacks_list = [checkpoint]

# Neural Network fitting and accuracy testing
NN_model.fit(X_train, y_train, epochs=1000,
validation_split = 0.2, callbacks=callbacks_list)
NNerror = r2_score(y_train, NN_model.predict(X_train))
print("Neural Network Training data:", NNerror)
Y_pred_2 = NN_model.predict(X_test)
nnerror = r2_score(y_test, Y_pred_2)
print("Neural Network Test data:", nnerror)

# Random Forest regression
rfregr = RandomForestRegressor(n_estimators=100,
n_jobs=-1, max_depth=12, verbose=True, random_state=1
```

```
, oob_score=True)
rfregr.fit(X_train, y_train)
rfrtrain = r2_score(y_train, rfregr.predict(X_train))
print("Random Forest Train data:", rfrtrain)
Y_pred_5 = rfregr.predict(X_test)
RFRerror = r2_score(y_test, Y_pred_5)
print("Random Forest Test data:", RFRerror)

# Gradient Boosting regression
gbregr = GradientBoostingRegressor(n_estimators=100,
max_depth=6, verbose=True, random_state=1)
gbregr.fit(X_train, y_train)
gbrtrain = r2_score(y_train, gbregr.predict(X_train))
print("Random Forest Train data:", gbrtrain)
Y_pred_6 = gbregr.predict(X_test)
GBRerror = r2_score(y_test, Y_pred_6)
print("Random Forest Test data:", GBRerror)

# ADA Boost regression
adaregr = AdaBoostRegressor(n_estimators=100,
learning_rate=0.9, random_state=1)
adaregr.fit(X_train, y_train)
adartrain = r2_score(y_train, adaregr.predict(X_train))
print("Random Forest Train data:", adartrain)
Y_pred_7 = adaregr.predict(X_test)
adaRerror = r2_score(y_test, Y_pred_7)
print("Random Forest Test data:", adaRerror)

# LightGBM regression
lgbmregr = lgb.LGBMRegressor(n_jobs=-1, subsample=1.0,
learning_rate=0.5, min_split_gain=.01)
lgbmregr.fit(X_train, y_train)
lgbmrtrain = r2_score(y_train, lgbmregr.predict(X_train))
print("Light GBM Train data:", lgbmrtrain)
lgbmrtrain = r2_score(y_train, lgbmregr.predict(X_train))
print("Light GBM Train data:", lgbmrtrain)
Y_pred_8 = lgbmregr.predict(X_test)
lgbmRerror = r2_score(y_test, Y_pred_8)
print("Light GBM Test data:", lgbmRerror)

```

```
# XGBoost regression
xgbregr = xgb.XGBRegressor(n_estimators=100, learning_rate
=0.2, max_depth=4)
xgbregr.fit(X_train, y_train)
XGBtrain = r2_score(y_train, xgbregr.predict(X_train))
print("XGBoost Regressor Train data:", XGBtrain)
Y_pred_9 = xgbregr.predict(X_test)
XGBerror = r2_score(y_test, Y_pred_9)
print("XGBoost Regressor Test data:", XGBerror)

pred1 = np.ravel(NN_model.predict(X_test))
pred2 = np.array(rfregr.predict(X_test))
pred3 = np.array(xgbregr.predict(X_test))
pred4 = np.array(lgbmregr.predict(X_test))
final_pred = np.mean(np.array([pred1, pred2, pred3,
pred4]), axis=0)
print(r2_score(final_pred, y_test))

# Reducing the number of features down to five based on feature
    ranking
features = ['TT', 'C', 'Cr', 'Mn', 'P']
X_final = data[features]
y_final = data[target]

# Splitting the new and reduced dataset into training and test data
Xtrain, Xtest, ytrain, ytest = train_test_split(X_final,
y_final, test_size=0.2, random_state=1)

# Neural Network
NN_model1 = Sequential()

# The Input Layer :
NN_model1.add(Dense(512, kernel_initializer='normal',
input_dim = 5, activation='relu'))

# The Hidden Layers :
NN_model1.add(Dense(256, kernel_initializer='normal',
activation='relu'))
NN_model1.add(Dense(128, kernel_initializer='normal',
activation='relu'))
```

```
NN_model1.add(Dense(64, kernel_initializer='normal',
activation='relu'))

# The Output Layer :
NN_model1.add(Dense(1, kernel_initializer='normal',
activation='linear'))

# Compile the network :
NN_model1.compile(loss='mean_absolute_error', optimizer=
'adam', metrics=['mean_absolute_error'])
NN_model1.summary()

# Neural Network fitting and accuracy testing
NN_model1.fit(Xtrain, ytrain, epochs=50, validation_split
= 0.2, callbacks=callbacks_list)
NNerror_1 = r2_score(ytrain, NN_model1.predict(Xtrain))
print("Neural Network Training data:", NNerror_1)
Y_pred_11 = NN_model1.predict(Xtest)
nnerror_1 = r2_score(ytest, Y_pred_11)
print("Neural Network Test data:", nnerror_1)

# Decision tree regressor
dtregr.fit(Xtrain, ytrain)
DTRtrain1 = r2_score(ytrain, dtregr.predict(Xtrain))
print("Decision Tree Train data:", DTRtrain1)
Y_pred_14 = dtregr.predict(Xtest)
DTRerror1 = r2_score(ytest, Y_pred_14)
print("Decision Tree Test data:", DTRerror1)

# XGBoost regressor
xgbregr.fit(Xtrain, ytrain)
XGBtrain1 = r2_score(ytrain, xgbregr.predict(Xtrain))
print("XGBoost Regressor Train data:", XGBtrain1)
Y_pred_12 = xgbregr.predict(Xtest)
XGBerror1 = r2_score(ytest, Y_pred_12)
print("XGBoost Regressor Test data:", XGBerror1)

# LightGBM regressor
lgbmregr.fit(Xtrain, ytrain)
lgbmrtrain1 = r2_score(ytrain, lgbmregr.predict(Xtrain))
```

```
print("LightGBM Train data:", lgbmrtrain1)
Y_pred_15 = lgbmregr.predict(Xtest)
lgbmRerror1 = r2_score(ytest, Y_pred_15)
print("LightGBM Test data:", lgbmRerror1)

# ADA Boost regressor
adaregr.fit(Xtrain, ytrain)
adartrain1 = r2_score(ytrain, adaregr.predict(Xtrain))
print("ADA Boost Train data:", adartrain1)
Y_pred_13 = adaregr.predict(Xtest)
adaRerror1 = r2_score(ytest, Y_pred_13)
print("ADA Boost Test data:", adaRerror1)

# Predicting the output using individual models of the ensemble and
    taking their mean to get the actual prediction from the ensemble
    model
final_pred = (np.ravel(NN_model1.predict(X_final)) +
np.array(dtregr.predict(X_final)) + np.array
(xgbregr.predict(X_final)) + np.array(lgbmregr.predict
(X_final)) + np.array(adaregr.predict(X_final)))/5

# Accuracy of Ensemble deep learning model on training and test data
Ensembletrain = r2_score(y_train, train_pred_10)
print("EnsembleDeep Learning model Train data:",
Ensembletrain)
Ensembleerror = r2_score(y_test, Y_pred_10)
print("Ensemble Deep Learning model Test data:",
Ensembleerror)
```

REFERENCES

1. **Abdi, H.** and **L. J. Williams** (2010). Principal component analysis. *WIREs Computational Statistics*, **2**(4), 433–459. ISSN 1939-5108. URL https://doi.org/10.1002/wics.101.

2. **Agrawal, A.**, **P. D. Deshpande**, **A. Cecen**, **G. P. Basavarsu**, **A. N. Choudhary**, and **S. R. Kalidindi** (2014*a*). Exploration of data science techniques to predict fatigue strength of steel from composition and processing parameters. *Integrating Materials and Manufacturing Innovation*, **3**(1), 8. ISSN 2193-9772. URL https://doi.org/10.1186/2193-9772-3-8.

3. **Agrawal, A.**, **P. D. Deshpande**, **A. Cecen**, **G. P. Basavarsu**, **A. N. Choudhary**, and **S. R. Kalidindi** (2014*b*). Exploration of data science techniques to predict fatigue strength of steel from composition and processing parameters. *Integrating Materials and Manufacturing Innovation*, **3**(1), 8. ISSN 2193-9772. URL https://doi.org/10.1186/2193-9772-3-8.

4. **Bessa, M.**, **R. Bostanabad**, **Z. Liu**, **A. Hu**, **D. W. Apley**, **C. Brinson**, **W. Chen**, and **W. Liu** (2017). A framework for data-driven analysis of materials under uncertainty: Countering the curse of dimensionality. *Computer Methods in Applied Mechanics and Engineering*, **320**, 633 – 667. ISSN 0045-7825. URL http://www.sciencedirect.com/science/article/pii/S0045782516314803.

5. **Breiman, L.** (2001). Random forests. *Machine Learning*, **45**(1), 5–32. ISSN 0885-6125. URL https://doi.org/10.1023/A:1010933404324.

6. **Castro, D.**, **J. Milan Ventura**, **C. Ruchert**, **D. Spinelli**, and **W. Wladimir Bose Filho** (2010). Influence of phosphorus content and quenching/tempering temperatures on fracture toughness and fatigue life of sae 5160 steel. *Materials Research*, **13**, 445–455.

7. **Chen, T.** and **C. Guestrin**, Xgboost: A scalable tree boosting system. *In Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16. ACM, New York, NY, USA, 2016. ISBN 978-1-4503-4232-2. URL http://doi.acm.org/10.1145/2939672.2939785.

8. **Chen, X.** and **J. C. Jeong**, Enhanced recursive feature elimination. *In Sixth International Conference on Machine Learning and Applications (ICMLA 2007)*, volume 1.

9. **Cheng, X.**, **B. Khomtchouk**, **N. Matloff**, and **P. Mohanty**, Polynomial regression as an alternative to neural nets. *In Computing research repository*. 2018. URL http://arxiv.org/abs/1806.06850.

10. **Dietterich, T. G.**, Ensemble methods in machine learning. *In Proceedings of the First International Workshop on Multiple Classifier Systems*, MCS '00. Springer-Verlag, London, UK, UK, 2000. ISBN 3-540-67704-6. URL http://dl.acm.org/citation.cfm?id=648054.743935.

11. **Dormehl, L.** (2019). What is an artificial neural network? here is everything you need to know. URL https://www.digitaltrends.com/cool-tech/what-is-an-artificial-neural-network/.

12. **Farrara, R.** and **J. H. Underwood** (1990). The effect of manganese phosphate coatings on fatigue crack initiation. *US Army Armament Research, Development and Engineering Center*, 28.

13. **Friedman, J. H.** (2002). Stochastic gradient boosting. *Comput. Stat. Data Anal.*, **38**(4), 367–378. ISSN 0167-9473. URL http://dx.doi.org/10.1016/S0167-9473(01)00065-2.

14. **Goodfellow, I.**, **Y. Bengio**, and **A. Courville**, *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

15. **Haftirman**, **S. Rahman**, **M. Anuar Mat**, and **F. Sutrisno**, Effect of tempering temperature on fatigue strength of thyssen 6582 steel. *In Conference on Applications and Design in Mechanical Engineering, Universiti Malaysia Perlis*, volume 55. 2007.

16. **Haykin, S.**, *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1998, 2nd edition. ISBN 0132733501.

17. **Ho, R.**, *Handbook of Univariate and Multivariate Data Analysis and Interpretation with SPSS*. Chapman & Hall/CRC, 2006. ISBN 1584886021.

18. **Idreos, S.**, **O. Papaemmanouil**, and **S. Chaudhuri**, Overview of data exploration techniques. *In Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD '15. ACM, New York, NY, USA, 2015. ISBN 978-1-4503-2758-9. URL http://doi.acm.org/10.1145/2723372.2731084.

19. **Kaelbling, L. P.**, **M. L. Littman**, and **A. P. Moore** (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, **4**, 237–285. URL http://people.csail.mit.edu/lpk/papers/rl-survey.ps.

20. **Ke, G.**, **Q. Meng**, **T. Finley**, **T. Wang**, **W. Chen**, **W. Ma**, **Q. Ye**, and **T.-Y. Liu**, Lightgbm: A highly efficient gradient boosting decision tree. *In Advances in Neural Information Processing Systems 30*. Curran Associates, Inc., 2017, 3146–3154. URL https://buildmedia.readthedocs.org/media/pdf/lightgbm/latest/lightgbm.pdf.

21. **Kvedaras, V.**, **J. Vilys**, **V. Ciuplys**, and **A. Ciuplys** (2006). Fatigue strength of chromium-plated steel. *Metallurgical and Materials Transactions A*, **12**.

22. **Liu, R.**, **A. Kumar**, **Z. Chen**, **A. Agrawal**, **V. Sundararaghavan**, and **A. Choudhary** (2015). A predictive machine learning approach for microstructure optimization and materials design. *Scientific reports*, **5**, 11551.

23. **Ma, Y.** and **G. Guo**, *Support Vector Machines Applications*. Springer Publishing Company, Incorporated, 2014. ISBN 3319022997, 9783319022994.

24. **Meyers, M. A.** and **K. K. Chawla**, *Mechanical Behavior of Materials*. Cambridge University Press, 2008, 2 edition.

25. **Mueller, T.**, **A. G. Kusne**, and **R. Ramprasad**, *Machine Learning in Materials Science*, chapter 4. John Wiley Sons, Ltd, 2016. ISBN 9781119148739, 186–273. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119148739.ch4.

26. **Murphy, K. P.**, *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012. ISBN 0262018020, 9780262018029.

27. **Ng, A.** (2016*a*). Activation functions. URL https://www.coursera.org/lecture/neural-networks-deep-learning/activation-functions-4dDC1.

28. **Ng, A.** (2016*b*). Backpropagation algorithm. URL https://www.coursera.org/lecture/machine-learning/backpropagation-algorithm-1z9WW.

29. **Quinlan, J. R.** (1986). Induction of decision trees. *Mach. Learn.*, **1**(1), 81–106. ISSN 0885-6125. URL http://dx.doi.org/10.1023/A:1022643204877.

30. **S. Hyde, R.**, **G. Krauss**, and **D. Matlock** (1994). Phosphorus and carbon segregation: Effects on fatigue and fracture of gas-carburized modified 4320 steel. *Metallurgical and Materials Transactions A*, **25**, 1229–1240.

31. **SUIMON, E.**, **T. NAKAMURA**, **H. OGUMA**, and **S. IKEDA** (2010). Effects of tempering temperature on very high cycle fatigue properties of sncm439. *Nihon Kikai Gakkai Ronbunshu, A Hen/Transactions of the Japan Society of Mechanical Engineers, Part A*, **76**, 440–442.

32. **Suresh, S.**, *Fatigue of Materials*. Cambridge University Press, 1998, 2 edition, 1–36.

33. **Tayanc, M.**, **A. Aytac**, and **A. Bayram** (2007). The effect of carbon content on fatigue strength of dual-phase steels. *Materials Design*, **28**, 1827–1835.

34. **Williams, C.** and **R. A. F. Hammond** (1954). The effect of chromium plating on the fatigue strength of steel. *Transactions of the IMF*, **32**(1), 85–106. URL https://doi.org/10.1080/00202967.1954.11869670.

35. **Yan, X.** and **X. G. Su**, *Linear Regression Analysis: Theory and Computing*. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 2009. ISBN 9789812834102, 9812834109.

36. **Zhu, J.**, **H. Zou**, **S. Rosset**, and **T. Hastie** (2009). Multi-class adaboost. *Statistics and its Interface*, **2**, 349–360. ISSN 1938-7989.